Filipe Miguel Santos de Oliveira

Licenciado em Ciência e Engenharia Informática

A Systems Approach to Searchable Encryption

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: Bernardo Luís da Silva Ferreira,
Professor Auxiliar,
Universidade de Lisboa

Co-orientador: João Carlos Antunes Leitão,
Professor Auxiliar,
Universidade NOVA de Lisboa

Júri

Presidente: Pedro Abílio Duarte de Medeiros
Arguente: Nuno Miguel Carvalho dos Santos

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Novembro, 2020

A Systems Approach to Searchable Encryption

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

À minha filha e esposa.

Agradecimentos

Ao Professor Bernardo Ferreira por todo aconselhamento, acompanhamento, disponibilidade e paciência que me dispensou ao longo destes últimos meses. O seu apoio e orientação, associados à liberdade para explorar novas ideias e soluções, foram fundamentais para a realização deste trabalho. Um agradecimento ao co-orientador, Professor João Leitão, que proporcionou o contacto com o Professor Bernardo e se disponibilizou para co-orientar este trabalho.

Um agradecimento muito especial à minha família, que fizeram um enorme esforço para que a conclusão do curso fosse possível. À minha filha Margarida e à minha esposa Sandra por toda a paciência e força que me transmitiram. Aos meus pais e irmãos pela ajuda e apoio que prestaram ao longo de todo o curso.

Ao Professor André Wemans, Tenente-Coronel Paulo Pires, Tenente-Coronel Henrique Azevedo, Major João Afonso, Major Bruno Ladeiro, Major Gonçalo Serrão, Major Alexis Vicente, Capitão Nelson Santos e Capitão Nuno Pragana por todo o apoio, aconselhamento e disponibilidade.

Aos colegas de faculdade que me acompanharam de forma mais próxima nestes últimos anos.

Este trabalho foi realizado com o apoio da FCT/MCTES, através do projeto estratégico NOVA LINCS (UIDB/04516/2020) e projeto HADES (PTDC/CCI-INF/31698/2017).

Muito obrigado a todos!

Resumo

O aumento da oferta de serviços na *cloud* veio facilitar o acesso a recursos de *hardware* e *software*. O número de clientes destes serviços tem vindo a aumentar e, com isso, aumentou o volume de dados que são armazenados nos servidores dos fornecedores. Contudo, muitos destes dados contêm informações que devem ser protegidas, por forma a garantir a privacidade dos seus proprietários.

A Criptografia Pesquisável (CP) e a Criptografia Simétrica Pesquisável (CSP) permitem, de forma eficiente, garantir a proteção dos dados e a capacidade de pesquisa dos mesmos, quando colocados em servidores dos fornecedores de serviços na *cloud*. No entanto, apesar dos vários trabalhos elaborados nesta área, alguns detalhes não são considerados ou são relegados para análises posteriores. Todavia, estes aspetos têm de ser estudados por forma a permitir a integração dos esquemas de CP com as operações sobre ficheiros reais.

Realizou-se a análise de alguns dos trabalhos mais recentes na área da CP e foram identificados os tópicos deixados em aberto na literatura mas que devem ser considerados quando se pretende desenvolver um sistema real que faça a manipulação de ficheiros. Estes tópicos dizem respeito ao armazenamento e operações realizadas sobre os ficheiros, custo financeiros, libertação de espaço dos índices, tratamento dos nomes originais dos ficheiros e suporte para múltiplas soluções de armazenamento.

Na sequência do trabalho de investigação, analisou-se a arquitetura tradicional dos esquemas de CP e desenhou-se uma nova arquitetura que não utiliza computação na *cloud*. Estas arquiteturas serviram de base à implementação de três sistemas que integram a manipulação de ficheiros reais com as operações do esquema de CP e que incorporam as propostas de solução para o tópico do armazenamento e operações sobre ficheiros, tratamento dos nomes dos ficheiros e operações de reindexação.

Os sistemas desenvolvidos foram avaliados no que respeita à sua performance, custos de operação e informações reveladas ao fornecedor de serviços.

Para além das soluções técnicas encontradas para os tópicos identificados no estudo, e que foram incorporadas nos sistemas, concluiu-se ainda que os sistemas testados apresentam vantagens em áreas distintas. O sistema que implementa a arquitetura tradicional cliente-servidor permite melhores performances na realização das operações enquanto

que o sistema que implementa a nova arquitetura e recorre à utilização de um *buffer* e de uma *cache* apresenta custos de operação mais baixos e revela menos informação ao fornecedor de serviços, conseguindo garantir a propriedade de *backward privacy*. O sistema que apenas utiliza um serviço de armazenamento revelou-se inadequado para uma utilização real, em virtude dos tempos elevados de inserção de elementos nos índices.

Palavras-chave: Criptografia Pesquisável, Criptografia Simétrica Pesquisável, *Cloud*, Privacidade, Segurança

Abstract

The expansion of cloud services facilitates access to hardware and software resources. The number of clients has been raising and so, the amount of data in the cloud provider's servers. Lots of this data have personal pieces of information that must be protected to guarantee privacy for data owners.

Searchable encryption (SE) and symmetric searchable encryption (SSE) provide an efficient way to protect data and enable searching operations when stored in cloud provider's servers. Although all the work in this area some details are left outside the scope or for future consideration. Is necessary to consider these details to integrate the searchable encryption schemes in real systems.

It was conducted a study about the most recent academic works in this field and found some points not considered in the literature. The identified topics have to be considered when manipulating files in operational systems and are related to file storage and operations, financial costs, reindex operations and file name transformation and multiple cloud support.

It was analysed the traditional architecture of searchable encryption schemes and was design a new one, that uses no cloud computation services. These two architectures were the base of the three implemented systems, which accomplish the integration of file handling with the searchable encryption scheme regarding file storage and file operations, filenames handling and reindex operations costs.

We accessed two of the three developed systems regarding performance, and all three regarding costs and security.

Beyond the technical solutions for the topics named in the research work, we concluded that accessed systems have advantages in different areas. The system with traditional client-server architecture is faster in search operations whereas the other, using buffer and cache, has lower operational costs and achieves better security, guaranteeing backward-privacy leakage. The system using only storage service revealed inadequate for real solutions, due to long times to insert index elements.

Keywords: Searchable Encryption, Symmetric Searchable Encryption, Cloud, Privacy, Security

Índice

Lista de Figuras

Lista de Tabelas

Glossário

Σοφος	Pronuncia-se "sophos" e é o nome do esquema de criptografia pesquisável criado por Raphaël Bost
backward privacy	Propriedade de um esquema de criptografia pesquisável ou criptografia simétrica pesquisável que garante que, na sequência de uma pesquisa, o fornecedor de serviços não consegue saber que documentos entretanto removidos continham a palavra que está a ser pesquisada.
cloud	Hardware e software de um ou vários *datacenter*
forward privacy	Propriedade de um esquema de criptografia pesquisável ou criptografia simétrica pesquisável que garante que, quando se efetua uma operação de *update*, esta operação não revela informação acerca das palavras-chave atualizadas.
leakage	Informação sobre os dados e sobre a sua utilização que é revelada ao servidor pela operação de um esquema de criptografia pesquisável ou criptografia simétrica pesquisável

Siglas

AEAD	*Authenticated Encryption with Associated Data*
AOF	*Append Only File*
API	*Application programming interface*
AWS	*Amazon Web Services*
BaaS	*Backend as a Service*
CDN	*Content Delivery Network*
CP	Criptografia Pesquisável
CSP	Criptografia Simétrica Pesquisável
DI	*Document Identification*
DP	*Document Presence*
EBS	*Elastic Block Store*
EC2	*Elastic Compute Cloud*
FaaS	*Function as a Service*
HDD	*Hard Disk Drive*
HMAC	*Hashed Message Authentication Code*
IaaS	*Cloud Infrastructure as a Service*
IDL	*Interface Definition Language*
IETF	*Internet Engineering Task Force*
NIST	*National Institute of Standards and Technology*
PaaS	*Cloud Platform as a Service*
PK	*Public Key*
PPR	*Partial Plaintext Recovery*

QR	*Query Recovery*
RCPRF	*Range-Constrained Pseudo Random Function*
RDB	*Redis Database Backup*
RU/s	*Request Units per Second*
S3	*Simple Storage Service*
SAA	Sistema Apenas Armazenamento
SAABC	Sistema Apenas Armazenamento com *Buffer* e *Cache*
SaaS	*Cloud Software as a Service*
SCA	Sistema Computação com Armazenamento
SDK	*Software Development Kit*
SE	*Searchable Encription*
SGBD	Sistema de Gestão de Bases de Dados
SO	Sistema Operativo
Sophos	*Scalable Optimal Forward Secure Searchable Encryption*
SSD	*Solid State Drive*
SSE	*Symmetric Searchable Encryption*
SST	*Sorted Sequence Table*
ST	*Search Token*
UT	*Update Token*
vCPU	*Virtual Central Processing Unit*
VM	*Virtual Machine*

Capítulo 1

Introdução

"There is no cloud, it's just someone else's computer"
Brian Greenberg - https://medium.com/@brian.greenberg.

1.1 Contexto e Motivação

Com o surgimento dos serviços disponibilizados na *cloud* foram criadas várias soluções que vieram ao encontro de uma multiplicidade de necessidades, tanto de clientes particulares como de empresas e organizações. Passou a ser possível disponibilizar recursos de computação, armazenamento ou aplicacionais de acordo com as necessidades e libertando os clientes de toda a sobrecarga que o planeamento, montagem e manutenção da infraestrutura obrigava.

Este modelo trouxe ainda vantagens financeiras, tanto para clientes como para fornecedores dos serviços. Os clientes passaram a conseguir aceder a recursos de forma mais económica e os fornecedores a tirar rendimento do investimento feito na montagem e manutenção das infraestruturas [1]. O conceito principal que possibilitou o desenvolvimento deste modelo foi a tecnologia de virtualização, permitindo alocar recursos a vários clientes em simultâneo e redistribuí-los sempre que necessário [1].

Contudo, ao utilizarem os serviços disponíveis na *cloud*, os clientes estão a confiar os seus dados ao fornecedor do serviço e, em alguns casos, muitos desses dados contêm informação pessoal que deve ser protegida por forma a garantir a privacidade. A solução para proteger os dados dos clientes passa por cifrar a informação [2], que apesar de aparentemente simples, traz desafios que têm necessariamente de ser resolvidos para que a manipulação e armazenamento dos dados seja feita de forma segura, eficiente e prática.

Uma solução imediata passaria por efetuar a cifra dos dados e colocá-los dessa forma no serviço de armazenamento na *cloud*. No entanto, esta proposta tem uma desvantagem que não torna a sua utilização apelativa: não é adequada para lidar com volumes de dados muito grandes e/ou cujo acesso a alguns documentos seja frequente, uma vez que qualquer ação de pesquisa ou processamento destes dados se torna impossível.

Existem soluções que não revelam nenhuma informação ao servidor, nomeadamente criptografia totalmente homomórfica ou esquemas de *oblivious RAM* [3] mas, até à data, nenhuma delas permite uma utilização eficiente, sendo mais lentas que a solução inicial de fazer o *download* de todos os documentos e realizar as pesquisas localmente no cliente [2].

A impraticabilidade destas soluções levou ao desenvolvimento de vários esquemas de Criptografia Pesquisável (CP) e Criptografia Simétrica Pesquisável (CSP) (cujos termos na língua inglesa são respetivamente *Searchable Encription* (SE) e *Symmetric Searchable Encryption* (SSE)), que possuem a eficiência necessária para se tornarem práticos e utilizáveis, permitindo a realização de pesquisas sobre os documentos cifrados [4]. Esta eficiência está acompanhada de um custo na medida em que a utilização do esquema revela informações sobre os dados armazenados ou *leaks* [3].

Na prática os esquemas de CP codificam informações do conteúdo de documentos numa estrutura de dados, que pode ser colocada e operada por um servidor na *cloud*. Isto possibilita libertar o cliente do armazenamento desta informação e realizar pesquisas sobre a estrutura de dados, ao mesmo tempo que garante privacidade dos dados e das pesquisas realizadas [5].

1.2 Problema

Os esquemas de CP permitem a eficiência necessária para realizar as operações de inserção, remoção e pesquisa sobre os índices [6] mas, para que seja obtida uma solução completa, utilizável e que permita salvaguardar a privacidade, é necessário considerar alguns aspetos de engenharia fora das construções teóricas do esquema escolhido, que permitam incorporar as operações sobre ficheiros.

Os vários trabalhos académicos desenvolvidos deixam em aberto algumas questões práticas, dado que o objetivo principal é a demonstração da validade e segurança do esquema de CP considerado. Assim, para que os esquemas e conceitos teóricos criados possam incorporar sistemas que manipulem ficheiros reais, estas questões têm de ser identificadas e analisadas, permitindo desenvolver soluções que aproveitem todo o potencial do esquema de CP escolhido.

Em suma, nesta dissertação pretendemos responder ao problema: *Quais os tópicos deixados em aberto pelos trabalhos académicos mais relevantes e recentes na área da criptografia pesquisável e, uma vez identificados, quais as soluções que permitirão a implementação de sistemas operacionais que realizem a manipulação de ficheiros, quais os seus custos e quais as informações reveladas ao fornecedor de serviços?*

1.3 Contribuições

As contribuições do trabalho realizado são:

- Apresentação dos tópicos deixados em aberto ou não considerados nos esquemas de CP, que resultam da investigação realizada com alguns dos trabalhos académicos mais recentes na área;
- Duas arquiteturas de sistemas de pesquisa de ficheiros cifrados baseados na *cloud*, uma baseada na arquitetura tradicional cliente-servidor e outra que não utiliza serviços de computação na *cloud*;
- Três sistemas que implementam as arquiteturas propostas e incorporam as soluções encontradas para os tópicos deixados em aberto pelos trabalhos académicos, designados por Sistema Computação com Armazenamento (SCA), Sistema Apenas Armazenamento (SAA) e Sistema Apenas Armazenamento com *Buffer* e *Cache* (SA-ABC). Os dois primeiros garantem *forward privacy* e o último garante a propriedade de *backward privacy*, superando o tipo de *backward privacy* com padrão de inserção;
- Análise do desempenho, modelo de custos e análise de segurança dos sistemas desenvolvidos.

1.4 Estrutura do Documento

O documento está organizado da seguinte forma: no capítulo 2 é abordado o estado da arte das tecnologias de *cloud* e esquemas académicos de CP, onde se analisa o surgimento e desenvolvimento do modelo de serviços de *cloud computing*, o início e desenvolvimento do trabalho na área da CP, os sistemas de bases de dados e, por último, os serviços de armazenamento na *cloud*. O capítulo 3 apresenta a primeira das contribuições desta dissertação, nomeadamente o resultado do trabalho de investigação realizado com alguns dos esquemas de CP mais recentes, sendo apresentados os tópicos deixados em aberto pelos trabalhos analisados. No capítulo 4 é apresentado o modelo de adversário, os conceitos teóricos do esquema de CP escolhido como base (o esquema *Scalable Optimal Forward Secure Searchable Encryption* (Sophos)) e as arquiteturas propostas. O capítulo 5 começa por apresentar o código base que foi utilizado no desenvolvimento dos sistemas e seguidamente detalha as alterações e extensões implementadas, terminando com a exposição das soluções técnicas encontradas para as questões listadas no capítulo 3. Por fim, o capítulo 6 apresenta as medidas de performance e resultados dos testes feitos aos sistemas, os modelos de custos e a avaliação de segurança.

Capítulo 2

Estado da Arte e Trabalho Relacionado

Neste capítulo será feita uma descrição do estado arte e trabalho relacionado. A secção 2.1 apresenta uma visão geral dos conceitos base da computação na *cloud* e a secção 2.2 apresenta a evolução dos esquemas de CP. Por fim, as secções 2.3 e 2.4 apresentam os serviços de bases de dados e serviços de armazenamento respetivamente.

2.1 Computação na Cloud

De acordo com o *National Institute of Standards and Technology* (NIST), a expressão *cloud computing* designa um modelo que permite o acesso ubíquo a um conjunto de recursos configuráveis de computação, partilhados e acedidos através da rede, de acordo com as necessidades dos clientes. Estes recursos caracterizam-se ainda pelo facto de estarem disponíveis para aquisição e utilização de forma relativamente simples e com interferência mínima do fornecedor do serviço [7].

2.1.1 Análise do Modelo

O modelo de computação na *cloud* possui cinco características essenciais, três modelos de serviço e quatro modelos de utilização (*deployment model*).

As características essenciais do modelo são respetivamente *On-demand self-service*, que estabelece que o cliente deve ser capaz de providenciar sozinho os recursos que necessita, sem ter de existir interação humana com o fornecedor de serviços, *broad network access*, que indica que os recursos deverão estar disponíveis para serem acedidos através da rede, *resource pooling*, que indica que o fornecedor deve conseguir satisfazer múltiplos clientes em simultâneo através de uma alocação dinâmica de recursos, quer físicos quer virtuais, *rapid elasticity*, que permite que os recursos sejam alocados e libertados pelos clientes

consoante as suas necessidades e, por fim *measured service*, que estabelece que este tipo de serviços deve permitir que a utilização dos recursos seja mensurável[7].

Os três modelos de serviço são o *Cloud Software as a Service* (SaaS), a *Cloud Platform as a Service* (PaaS) e a *Cloud Infrastructure as a Service* (IaaS) [7]. Apesar de possuirem nomes distintos, não possuem fronteiras totalmente disjuntas, existindo serviços que podem pertencer a mais que um modelo de serviço [8]. Cada um destes modelos é caracterizado da seguinte forma [7]:

- SaaS: modelo em que o cliente tem ao seu dispor aplicações criadas pelo fornecedor de serviços e cujo funcionamento é suportado pela infraestrutura da *cloud*. O cliente não tem controlo sobre quaisquer recursos, conseguindo fazer apenas algumas configurações próprias da aplicação. O *Google Docs*, *Sheets* e *Slides* são exemplos deste modelo;
- PaaS: modelo que permite ao cliente colocar as suas aplicações na infraestrutura, aplicações essas desenvolvidas com as tecnologias suportadas pelo fornecedor do serviço. À semelhança do que acontece no modelo anterior, o cliente não gere quaisquer recursos da infraestrutura. A *Google App Engine* é um exemplo deste modelo;
- IaaS: modelo que fornece ao cliente recursos computacionais e que permite controlar todas as camadas de *software* que são executadas nesses recursos. Um exemplo deste modelo é o serviço *Elastic Compute Cloud* (EC2) da *Amazon Web Services* (AWS).

Os modelos de utilização são quatro, nomeadamente a *cloud* privada, a *cloud* pública, a *community cloud* e a *cloud* híbrida [7]. Armbrust et al. [8] definem como *clouds* públicas aquelas cujo acesso está aberto ao público de maneira geral e como *clouds* privadas as que são constituídas por *data centers* proprietários de uma determinada empresa ou organização e, como consequência, não é possível a sua utilização de forma pública. As *community clouds* dizem respeito às infraestruturas de *cloud* que são utilizadas em exclusivo por comunidades específicas de utilizadores que partilham interesses comuns e, por último, as *clouds* híbridas que designam infraestruturas compostas por uma combinação de dois ou mais modelos de implementação [7].

2.1.2 *Serverless Computing*

2.1.2.1 Caracterização

Recentemente surgiu o modelo de *serverless computing*. Jonas et al. [9] estabelecem que para um serviço ser considerado *serverless* deve possuir obrigatoriamente duas características, nomeadamente a capacidade de escalar de forma automática sem necessidade de provisionamento explícito e ser cobrado ao cliente com base na utilização e não na quantidade de recursos. Ainda segundo os mesmos autores, este modelo resulta da junção de dois modelos de serviço, respetivamente *Function as a Service* (FaaS) e *Backend as a Service* (BaaS), definidos da seguinte forma:

- FaaS: designa o tipo de serviços na *cloud* onde o cliente apenas escreve o código que pretende que seja executado e todas as restantes tarefas de provisionamento de recursos e gestão ficam a cargo do fornecedor do serviço. Como exemplos deste modelo temos os serviços AWS Lambda [10], Google Cloud Functions, IBM Cloud Functions ou Azure Functions [9];

- BaaS: *serverless framework* especializada para responder a requisitos específicos das aplicações como *serverless databases* ou *serverless big data processing frameworks*. Como exemplo deste modelo podemos citar o Amazon *Simple Storage Service* (S3) [9, 10].

Existem ainda três diferenças principais entre o *serverless computing* e o *serverful computing* [9]:

- Desacoplamento de computação e armazenamento: que devem escalar de forma independente e, como tal, cobrados separadamente. A computação não tem estado e o armazenamento é garantido por um outro serviço na *cloud*;

- Código executado sem gestão da alocação de recursos: o cliente apenas fornece o código que deseja executar, ficando a cargo do fornecedor do serviço todas as tarefas necessárias para ser executado com sucesso;

- Pagamento por recursos usados e não por recursos alocados: o pagamento está associado a medidas de utilização como o tempo durante o qual o código está a executar em vez se cobrar por capacidade de computação ou número de *Virtual Machines* (VMs) alocadas.

2.1.2.2 *Cloud Functions*

De uma forma geral, uma *cloud function* é escrita numa linguagem de alto nível suportada pelo fornecedor do serviço e à qual está associado um evento que provoca a execução do código. Esta estrutura simplifica o desenvolvimento de aplicações e torna os recursos da *cloud* mais fáceis de utilizar [9].

As *cloud functions* apresentam capacidades de escalamento automático muito boas, sendo esta a característica que suporta o modelo de pagamentos por recursos usados, pois um controlo mais fino e rápido da alocação de recursos permite um modelo de cobrança mais granular [9].

As capacidades de isolamento, por sua vez, permitem uma distribuição dos recursos de hardware pelo vários clientes de forma eficiente, rentável e segura [9]. No entanto, para conseguir o grau de isolamento referido ao mesmo tempo que se conseguem escalamentos rápidos, não permitidos pelas VMs por si só, são empregues técnicas que permitem contornar essa limitação. Um dos exemplos das técnicas citadas são as utilizadas pelo serviço AWS Lambda, que mantém duas *pools* de recursos disponíveis: uma *"warm pool"*

de VMs prontas a serem atribuídas a um cliente e uma *"active pool"* de VMs que acabaram de executar código e estão prontas para serem invocadas novamente [9].

2.1.2.3 *Edge Computing*

Segundo Jonas et al. [9] o *serverless computing* vai transformar também o *edge computing*, referindo que vários *Content Delivery Network* (CDN) já oferecem a possibilidade de executar *serverless functions* em localizações mais próximas dos utilizadores, como o serviço Lambda@Edge da AWS [11], e até mesmo nos dispositivos, como o serviço AWS Greengrass [12].

2.1.3 Vantagens e Desvantagens

O modelo de computação na *cloud* trouxe consigo um conjunto de conceitos inovadores pois permitiu transformar a computação num serviço, contratado de forma simples pelos clientes de acordo com as suas necessidades e providenciando uma enorme elasticidade nos recursos. Fornece a ilusão de recursos infinitos a que os clientes podem aceder sem provisionamento prévio, eliminando a obrigatoriedade de fazer previsões e compromentimentos antes de se conhecerem as necessidades reais, e um novo modelo de negócio que oferece a possibilidade de se pagar pelos serviços com base nos recursos que se utilizam e não nos que estão alocados [1]. Com o surgimento do modelo *serverless computing* a utilização da *cloud* tornou-se mais simples, o que permitiu que novos clientes passassem a utilizar os serviços e que os clientes que já utilizavam conseguissem passar a tirar maior rendimento ou utilizar novos recursos, beneficiando de aumentos de produtividade e, eventualmente, de redução de custos de utilização [9].

No entanto este novo modelo não está livre de alguns pontos menos positivos ou obstáculos que têm que ser considerados, como referido por Armbrust et al. [1]. Um dos aspetos mencionados e importante no âmbito do tema deste trabalho é designado por *"Data confidentiality e auditability"*, uma vez que a segurança é um dos pontos negativos mais citados acerca da utilização dos serviços de *cloud*. Apesar das questões de segurança não se esgotarem na proteção dos dados dos utilizadores, sendo esta uma área bastante mais vasta, neste ponto em específico é apontada a cifra dos dados como uma solução em consequência de se estar a confiar os dados à empresa prestadora dos serviços [1].

2.2 Criptografia Pesquisável

Nesta secção será apresentada a evolução dos esquemas de CP, com referência para os trabalhos considerados mais relevantes na área. Desde o primeiro artigo publicado [13] foram apresentados vários esquemas de CP, onde estavam presentes diferentes características e formas de resolver os problemas. Uma das evoluções apresentadas foi dinamicidade, passando os esquemas a permitir a adição e remoção de ficheiros à base de dados [14] evitando a cifra de todos os documentos logo numa fase inicial [2].

Os esquemas de CP dinâmicos trouxeram novas questões de segurança. Assim, como sumarizado por Chamani et al. [15], a adição de novos ficheiros à base de dados revela que o documento inserido tem palavras-chave que já foram pesquisadas anteriormente e as pesquisas por uma determinada palavra-chave podem revelar os ficheiros que, no passado, estiveram na base de dados e que a continham. Foram desenvolvidos esquemas que permitiram evitar a revelação destas informações para os servidores e são analisados nas subsecções 2.2.4 e 2.2.5.

2.2.1 Esquemas Estáticos

O primeiro esquema de CSP proposto foi apresentado no ano 2000, no trabalho de Song et al. [13] [16]. A proposta apresentada ambicionava solucionar a questão da pesquisa sobre dados cifrados através de um esquema não-interactivo, com tempo de pesquisa linear no tamanho da coleção de documentos [16]. Este esquema caracteriza-se por utilizar *probabilistic searching*, em que a pesquisa por uma determinada palavra-chave retorna não só todos os documentos onde a palavra surge mas também alguns documentos onde ela não está contida [13].

Mais tarde, Curtmola et al. [4] apresentam o primeiro esquema de CSP que permitia obter tempos de pesquisa sublineares no número de documentos guardados na base de dados. O modelo de segurança foi também melhorado e introduziram a noção de segurança adaptativa para os esquemas de CSP [5].

A grande desvantagem dos esquemas estáticos está na obrigatoriedade de cifrar todos os documentos antes de serem colocados nas soluções de armazenamento na *cloud*, o que se revela um problema para coleções com um elevado número de documentos, e por não ser possível a adição subsequente de novos documentos, tendo de se repetir a cifra de toda a base de dados [16].

2.2.2 Esquemas Dinâmicos

Para que os esquemas de criptografia pesquisável pudessem ser incorporados em soluções reais, como parte integrante de *software* funcional, eficiente e seguro, era importante o desenvolvimento de esquemas dinâmicos. Um esquema dinâmico caracteriza-se por permitir que o utilizador possa fazer a inserção e remoção de documentos do conjunto de documentos cifrados sem ter de realizar operações que afetem toda a coleção [16].

Kamara et al. [16] criaram um esquema dinâmico e definiram as características que estes tipos de esquemas deveriam possuir para que fosse possível a sua utilização em sistemas reais. Este trabalho tinha, contudo, alguns problemas de *leakage* tendo sido mais tarde proposta uma nova abordagem [17]. O novo esquema apresentava a vantagem de permitir que as pesquisas fossem paralelizáveis através da eliminação das ineficiências resultantes da utilização de índices invertidos. A solução proposta fazia uso de uma árvore *red-black* que, segundo os autores, tornava o esquema mais eficiente [17].

Posteriormente, Cash et al. [18] apresentam um novo esquema dinâmico que possuía como principal vantagem a escalabilidade, passando a permitir uma utilização eficiente em conjuntos de dados de grandes dimensões.

2.2.3 Ataques

Foram realizados trabalhos que estudaram os esquemas de CP de um ponto de vista prático [3, 6, 19] e mostraram que os *leakages* revelados poderiam ser utilizados para obter informação acerca do conjunto de dados. No entanto, a possibilidade de realizar esses ataques não contradiz as garantias de segurança dos esquemas em causa, revela apenas que ao ser conhecida determinada quantidade de informação aparentemente sem utilidade, esta pode ser explorada por atacantes [20].

2.2.3.1 Modos de Ataque, Conhecimento e Objetivos

Os trabalhos referidos em 2.2.3 podem ser agrupados relativamente ao modelo de ataque que consideraram. O modelo de ataque é composto por modos de ataque, conhecimento que o adversário possui acerca dos documentos e os objetivos do ataque [6].

Modos de Ataque O trabalho realizado por Cash et al. [6] define três modos de ataque, que estão relacionados com a capacidade do servidor adversário realizar ataques ativos ou passivos:

- Ataques realizados por um servidor *honest-but-curious*, que executa o protocolo estabelecido de forma correta mas que tenta obter informação acerca do texto original que compõe os documentos cifrados ou sobre as *queries* que são feitas sobre os dados durante as pesquisas;
- *Chosen-document attacks*, onde o adversário consegue fazer com que o cliente do sistema introduza documentos escolhidos por si;
- *Chosen-query attacks*, onde o adversário consegue levar o cliente do sistema a realizar *queries* escolhidas por si e tirar proveito da informação que o esquema revela na sequência dessas *queries*.

Conhecimento do Adversário Ainda no mesmo trabalho [6] é referido que o conhecimento que o servidor possui acerca dos documentos que tem armazenados pode potenciar a extração de informação do esquema CSP que está a ser utilizado. São considerados os seguintes tipos de conhecimento do adversário:

- *Distributional query knowledge*, que engloba os casos onde o servidor tem indícios do tipo de *queries* que vão ser realizadas sobre os dados armazenados, em virtude de conhecer a sua natureza;

- *Known queries*, que diz respeito aos casos em que o servidor conhece os termos ou *keywords* que vão ser pesquisados nos documentos.
- *Distributional document knowledge*, que engloba os casos em que o servidor dispõe de conhecimento do tipo de documentos que estão cifrados e, nesse caso, consegue saber que tipo de dados e padrões se espera que estejam contidos nesses documentos;
- *Known documents*, que refere os cenários em que o servidor atacante conhece o *plaintext* de alguns documentos cifrados ou, não tendo esses dados, tem muita informação significativa sobre alguns desses documentos.
- *Fully-known document set*, que refere os casos em que o atacante tem conhecimento de todos os documentos e apenas desconhece as *queries* que são realizadas pelo cliente (total ou parcialmente).

Objetivos dos Ataques No que respeita aos objetivos dos ataques, Cash et al. [6] definem:

- *Query Recovery* (QR) define o objetivo de determinar o *plaintext* das consultas efetuadas pelo cliente sobre os documentos armazenados;
- *Partial Plaintext Recovery* (PPR) que define o objetivo de reconstruir os documentos que estão armazenados;
- *Document Presence* (DP) que define os ataques que pretendem determinar se um ou alguns determinados documentos, dos quais se conhece o *plaintext*, se encontram no índice de documentos do cliente;
- *Document Identification* (DI) cuja intenção é encontrar os identificadores dos documentos, dados pelo esquema de CSP, com base nos documentos que são do conhecimento do atacante.

2.2.3.2 O primeiro estudo sobre ataques

O início dos estudos experimentais sobre a segurança dos esquemas de CP foi realizado por Islam et al. [19]. O ataque experimentado foi, de acordo com a categorização apresentada em 2.2.3.1, um *QR attack* com conhecimento total do conjunto de documentos e conhecimento parcial das *queries* realizadas [6]. Com este ataque demonstrou-se que, se o atacante tivesse na sua posse o conhecimento referido, conseguia determinar quais as pesquisas efetuadas pelo cliente [3].

Um dos pontos fortes deste ataque é que a taxa de sucesso é independente do número de pesquisas realizadas, o que permite que com uma quantidade razoável (de pesquisas) se consigam realizar ataques eficazes [6]. Por outro lado, este tipo de ataque apresenta

duas desvantagens. A primeira está relacionada com a incapacidade de lidar com conjuntos de palavras-chave superiores a 5000, revelando que não é escalável, e a segunda está relacionada com a dependência que o ataque tem em relação às pesquisas que o atacante conhece pois, face a aumentos das palavras-chave, se as pesquisas conhecidas forem poucas, a taxa de recuperação é baixa ou até mesmo zero [6].

Desta forma, e relaxando a quantidade de documentos conhecidos pelo atacante, Cash et al. [6] demonstraram que o ataque realizado por [19] não apresenta bons resultados nem é realista, dada a quantidade de informação que o atacante tem de conhecer *à priori* sobre os dados e também devido à complexidade da solução que foi encontrada e que permite recuperar as *queries* feitas [6].

2.2.3.3 Exploração dos Ataques Ativos e Necessidade de *Forward Privacy*

Cash et al. [6] realizaram um ataque semelhante ao apresentado no trabalho de Islam et al. [19], um ataque de QR. No entanto, apesar do trabalho de Cash et al. [6] ter revelado uma eficácia superior, o seu sucesso estava dependente do servidor conhecer o número de documentos que continham determinada palavra-chave.

No que respeita aos ataques de PPR, são realizados e analisados dois tipos de ataque passivo e, pela primeira vez, explorados os riscos de ataques ativos em que o atacante consegue fazer com que o cliente insira documentos escolhidos por si na base de dados cifrada [6].

No trabalho desenvolvido por Zhang et al. [3] é aprofundado o estudo das consequências da revelação de informações pelos esquemas de CP no que diz respeito aos *file injection attacks* (ou *chosen document attack* de acordo com a classificação de [6]), isto é, ataques em que o servidor comprometido consegue escolher ficheiros e enviá-los ao cliente, de forma a que sejam tratados e armazenados pelo esquema de CP que está a ser utilizado. São considerados os ataques adaptativos e os não adaptativos. No entanto, se os esquemas de CP utilizados garantirem a propriedade de *forward privacy*, os ataques adaptativos não surtem efeito [3]. Os esquemas com esta propriedade evitam que o servidor perceba se um documento que está a ser inserido pela primeira vez na base de dados possui palavras-chave pesquisadas anteriormente [3].

As conclusões deste estudo salientam a importância dos esquemas de criptografia pesquisável garantirem *forward privacy* mas adiantam que estes ataques não são importantes se, no cenário em que é utilizado o esquema de CP, os *file injection attacks* não são uma preocupação [3]. No entanto, e apesar desta opinião, outros investigadores defendem que se deve sempre tentar minimizar a informação que o servidor consegue aprender [5].

Em suma, o trabalho de Zhang et al. [3] demonstrou que o *leakage* dos padrões de acesso aos ficheiros é perigoso e indica que será necessário que futuros esquemas garantam *forward privacy* [3].

2.2.4 Esquemas *Forward Private*

Em 2016, aquando da publicação do trabalho realizado por Raphaël Bost [2], os únicos esquemas de CSP existentes e que garantiam *forward privacy* eram os esquemas de Chang et al. [21] e o de Stefanov et al [14], mas ambos eram ineficientes [2]. Bost apresenta o seu esquema de CP que garante *forward privacy*, com complexidade ótima nas pesquisas e nas atualizações de documentos.

Mais tarde Bost et al. [20] apresentaram um esquema de CP mais eficiente, utilizando *constrained* e *puncturable pseudo-random functions*. O esquema, designado por "Diana", consegue performances dez vezes superiores quando comparado com esquemas de CP com o mesmo grau de *leakage* [20].

2.2.5 Esquemas *Forward and Backward Private*

Os esquemas mais recentes reduzem a quantidade de informação revelada servidor, uma vez que garantem a propriedade de *backward privacy*. Esta propriedade garante que as pesquisas realizadas pelo cliente não revelam informações sobre as entradas que correspondem à pesquisa e que foram, entretanto, eliminadas da base de dados, isto é, limita a informação das operações de *update* que o servidor consegue aprender, na sequência da realização de uma pesquisa. [20].

No trabalho de Bost et al. [20] é apresentada, pela primeira vez, uma definição formal de *backward privacy*. Os três tipos de *leakage* podem ser definidos da seguinte forma [15]:

- **Tipo I** (*Backward privacy* com padrão de inserção): os esquemas de tipo I revelam, durante uma pesquisa com uma determinada palavra-chave w, o número e tipo de atualizações que estão associadas a essa palavra-chave, os identificadores dos ficheiros que atualmente contêm a palavra-chave e estão na base de dados e, por fim, quando a inserção do ficheiro foi realizada;
- **Tipo II** (*Backward privacy* com padrão de atualizações (ou *updates*)): para além das informações reveladas no tipo I, são revelados ainda o momento em que foram realizadas atualizações correspondentes a uma palavra-chave w;
- **Tipo III** (*Backward privacy* fraca): estes esquemas revelam ainda que atualização de remoção, ou *delete*, cancelou uma inserção anterior.

Ainda no mesmo trabalho [20] são apresentados vários esquemas de CP. São descritos dois esquemas genéricos que mostram como se pode obter um esquema com *backward privacy* a partir de um esquema *forward private*, a troco de uma interação de comunicação extra por cada pesquisa realizada pelo cliente. As instanciações dos esquemas genéricos foram nomeadas pelos autores como "Moneta", que garante *backward privacy* do tipo I, e "Fides", do tipo II. É também proposto um esquema, designado por "Diana$_{del}$", uma modificação do esquema "Diana" já referido em 2.2.4 e que garante *backward privacy* do tipo III e, ainda, proposto o "Janus", também do tipo III. Os esquemas "Fides", "Diana$_{del}$"

e "Janus" são os primeiros, não baseados em técnicas de *oblivious RAM*, a garantir a propriedade de *backward privacy* e, o esquema "Janus" é o único esquema que garante *backward privacy* utilizando apenas um *single-roundtrip* [20].

Chamani et al. [15] apresentaram mais tarde novos esquemas com garantias de *forward privacy* e *backward privacy* designados "Mitra", "Orion" e "Horus". O esquema "Mitra" garante *backward privacy* tipo II e apresenta uma performance semelhante à construção "Fides" mas melhores tempos de computação nas pesquisas e atualizações, o que permite ser caracterizado pelos autores como esquema CSP *forward private* e *backward private* mais eficiente [15]. O esquema "Orion" garante *backward privacy* do tipo I e, por sua vez, o esquema "Horus" garante *backward privacy* do tipo III. O esquema "Horus" apresenta melhores performances nas pesquisas e menos rondas de interação, o que pode tornar este esquema uma boa opção quando a performance for o principal objetivo [15].

2.3 Sistemas de Bases de Dados

2.3.1 Bases de Dados na Cloud

2.3.1.1 Amazon DynamoDB

O Amazon DynamoDB é um serviço de base de dados NoSQL da AWS, que dá garantias de performance e escalabilidade, libertando os clientes das preocupações com configurações, replicação, *software patching* ou escalamento. Suporta dois modelos distintos, o modelo chave-valor e modelo documental [22].

Os componentes principais do DynamoDB são tabelas, items e atributos. Uma tabela é uma coleção de items e cada item é, por sua vez, uma coleção de atributos. Os items em cada tabela possuem um identificador único, a *primary key*, e existe suporte para índices secundários, limitados a vinte *global secondary indexes* e cinco *local secondary indexes* por tabela [22].

O Amazon DynamoDB está disponível em várias regiões AWS independentes. Cada uma destas regiões é composta por várias *availability zones* ligadas entre si, garantindo baixas latências na comunicação mas independentes no que respeita a falhas [22].

Este sistema de base de dados fornece dois tipos de consistência: eventual (por defeito) e forte, este último com algumas desvantagens [22].

Os dados são armazenados em partições, termo utilizado para referir a alocação de recursos de armazenamento para as tabelas, apoiados em *Solid State Drives* (SSDs), e são replicados de forma automática por várias *availability zones*, dentro da mesma região AWS [22].

Estão disponíveis dois modos para processamento das leituras e escritas sobre as tabelas. Cada um tem uma forma diferente de ser cobrado aos clientes e uma forma diferente de gerir a capacidade da base de dados para responder aos pedidos [22]:

- ***On-Demand mode***: modelo de cobrança mais flexível. Permite processar milhares

de pedidos por segundo sem necessidade de planeamento prévio, onde o cliente paga por cada pedido feito à base de dados. A base de dados ajusta-se para fazer face a picos de carga, pelo que este modo é adequado quando se desconhecem as quantidades de acessos, necessidades da aplicação ou quando o modelo de pagamento é mais vantajoso;

- ***Provisioned mode***: neste modelo o cliente indica a quantidade de leituras e escritas que pretende para a aplicação. Permite prever os custos de utilização.

2.3.1.2 Microsoft Azure Cosmos DB

A Azure Cosmos DB é uma base de dados multi-modelo, globalmente distibuída. Permite obter baixas latências, elevada escalabilidade, vários níveis de consistência e alta disponibilidade [23].

Os dados são geridos criando *databases*, *containers* e items. Os *containers*, dependendo da *Application programming interface* (API) escolhida, podem ser instanciados como *collection*, *table* ou *graph*. Agrupam items que, também de acordo com a API escolhida, podem ser instanciados como *document*, *row*, *node* ou *edge* [23].

À semelhança do que acontece no serviço DynamoDB, também a Azure CosmosDB replica os dados pelas várias *Azure Regions*. Dentro de cada região existem partições físicas e todas as partições possuem um conjunto de réplicas [23]. Quando uma escrita é comunicada ao cliente como bem sucedida significa que os dados estão armazenados com sucesso num *quorum* de réplicas dentro da região que aceitou as escritas. Desta forma, para garantir a disponibilidade dos dados em caso de falha de uma região, a documentação [23] aconselha a ter pelo menos duas regiões configuradas.

Uma das vantagens que este sistema apresenta são os vários tipos de consistência que são oferecidos ao cliente, permitindo uma escolha mais adequada às necessidades da aplicação, em termos de disponibilidade e performance. Os cinco níveis de consistência disponíveis, do modelo mais forte para o mais fraco, são: *Strong consistency*, *Bounded Staleness*, *Session*, *Consistent prefix* e *Eventual consistency* [23].

A Azure CosmosDB é cobrada com base em *Request Units per Second* (RU/s), sendo que o provisionamento mínimo para cada *database* ou *container* é de 400 RU/s (100 RU/s correspondem a 267,8 milhões de leituras por mês), sendo o valor a pagar multiplicado pelo número de regiões pretendidas [23].

2.3.1.3 Google Cloud Firestore

Este Sistema de Gestão de Bases de Dados (SGBD) é um sistema NoSQL da Google que utiliza o modelo documental [24]. É o sucessor da Cloud Datastore, pertencente à mesma empresa [25].

Armazena os dados como documentos, organizados em coleções. Cada documento é constituído por conjuntos de pares chave-valor e pode conter ainda sub coleções e objetos

compostos (*nested objects*), que por sua vez podem incluir campos com tipos de dados primitivos ou complexos (i.e. listas). O sistema está otimizado para armazenar coleções de grandes dimensões mas compostas por pequenos documentos [24].

O utilizador tem disponíveis duas opções de localização para os dados, a localização multi-região e a localização regional. Quando é escolhida a opção multi-regional, os dados são replicados em várias regiões dentro da mesma área geográfica e, dentro de uma região, os dados são replicados por várias zonas. Desta forma, para obter a máxima disponibilidade e tolerância a falhas deve ser utilizada a opção multi-regional e para conseguir utilizar o sistema com custos mais reduzidos deve ser escolhida a opção regional [24].

Por ser a geração seguinte da antiga Google Cloud Datastore, quando é criada uma nova base de dados, estão disponíveis dois modos, o *native mode* e o *datastore mode*, este último para garantir compatibilidade com aplicações que fizessem uso da Cloud Datastore [24]. O sistema garante consistência forte nos dois modos de operação mas oferece diferentes performances nas escritas, que têm de ser consideradas de acordo com a solução que se está a desenvolver. A documentação oficial sugere que se utilize o *datastore mode* para novos projetos de servidores, pois este modo possibilita milhões de escritas por segundo, e que o *native mode* seja utilizado para novas aplicações móveis e aplicações web pois permite atender milhões de clientes em simultâneo [24].

O modelo de pagamentos deste sistema está dependente do número de leituras, escritas e remoções que são efetuadas, da quantidade de espaço de armazenamento que os dados ocupam e da quantidade de tráfego utilizado. Tanto as operações realizadas (leituras, escritas e remoções) como o armazenamento são cobrados com valores diferentes dependendo da localização, sendo fixos os valores das cotas diárias gratuitas. Em relação ao tráfego, este é cobrado com diferentes valores consoante tenha origem dentro da Google Cloud Platform ou não e dependendo da quantidade e localização de saída dos dados [24].

2.3.2 Bases de Dados *In-memory*

2.3.2.1 Redis

O Redis é uma solução *open source* para armazenamento de dados em memória, que pode ser utilizada como base de dados, *cache* ou sistema de distribuição de mensagens (*message broker*) [26]. É caracterizada na documentação como representando um *trade-off* entre a possibilidade de efetuar leituras e escritas muito rapidamente mas com a desvantagem dos conjuntos de dados não poderem ser de dimensões superiores à dimensão da memória. Oferece persistência através da realização de escritas em disco [26].

De acordo com a documentação oficial [26], este sistema é um servidor de estruturas de dados com suporte para diferentes tipos de valor, ao invés de ser uma normal base de dados chave-valor. Por esta razão não permite apenas associar chaves do tipo *string* a valores do mesmo tipo mas também a estruturas de dados mais complexas, como

são exemplo listas, conjuntos e conjuntos ordenados. Permite ainda utilizar como chave qualquer sequência binária [26].

No que à persistência diz respeito, são oferecidas quatro possibilidades [26]: a persistência *Redis Database Backup* (RDB), que em períodos de tempo especificados guarda *snapshots* dos dados, a persistência *Append Only File* (AOF), que escreve um registo com todas as operações de escrita que são solicitadas ao servidor e que permite reconstruir a base de dados quando o servidor é reiniciado. Existe a possibilidade de não ser utilizada nenhuma forma de persistência, estando os dados disponíveis apenas durante a execução do servidor e, por último, é possível combinar os dois primeiros modos, AOF e RDB.

A opção RDB apresenta como vantagem o facto de ser um único ficheiro que guarda toda a representação dos dados. Isto torna esta opção vantajosa enquanto ficheiros de *backup* e de histórico da base de dados. Por ser um ficheiro compacto pode facilmente ser transferido entre *data centers* e possibilita reinícios mais rápidos, quando comparado com a opção AOF. A sua principal desvantagem é a perda de dados escritos recentemente pois, em caso de falha, toda a informação escrita desde o último *snapshot* não é recuperada [26].

A opção AOF, por sua vez, permite um controlo mais fino da forma como se garante a durabilidade dos dados, através de uma política de sincronização, tornando menores as perdas dos últimos dados escritos. A política de sincronização, por defeito, estabelece que são colocados em armazenamento persistente todas as alterações a cada segundo. Utiliza ainda um registo onde todas as operações efetuadas são colocadas e um sistema que permite lidar com as situações em que o registo se torna demasiado grande. As desvantagens desta opção são o facto dos ficheiros utilizados possuirem maiores dimensões que os ficheiros da opção RDB e o facto de ser mais lenta, dependendo da política de sincronização escolhida [26].

Existe a possibilidade de configurar a replicação da base de dados, no modelo primário-secundário. É utilizada replicação assíncrona por defeito e a replicação é não bloqueante para o primário, que continua a responder a *queries* durante operações de sincronização das réplicas. A replicação pode ser utilizada para garantir a escalabilidade da aplicação (podendo dispor de multiplas réplicas para operações de leitura) como para garantir elevada disponibilidade dos dados [26].

Os fornecedores de serviços na *cloud* disponibilizam soluções que permitem libertar os clientes das questões relacionadas com a inicialização, execução e gestão das bases de dados *in-memory*, oferecendo soluções compatíveis com o redis, permitindo a sua utilização sem alterar as aplicações que já faziam uso desta tecnologia e tirando partido da disponibilidade e fiabilidade que garantem. Como exemplo destes serviços temos a AWS ElastiCache [27], Azure Cache for Redis [28] e Google Cloud Memorystore [29].

2.3.2.2 RocksDB

O RocksDB [30] é um sistema *open source* de bases de dados chave-valor onde, quer as chaves quer os valores a elas associado, são *streams* de *bytes* de tamanho arbitrário.

Suporta diferentes tipos de *hardware* de armazenamento, nomeadamente memória RAM, memória *flash*, *Hard Disk Drive* (HDD) ou armazenamento remoto, mas foi desenvolvida com o objetivo principal de utilizar armazenamento *flash* [31].

Os três componentes base deste sistema são [31]:

- *Memtable*: estrutura de dados em memória, onde são inseridas as novas escritas realizadas e, opcionalmente poderão ser escritas para o ficheiro de *log*;
- Ficheiro *Sorted Sequence Table* (SST): ficheiro em armazenamento persistente para onde é descarregada a informação da *memtable* quando esta atinge o seu limite máximo de armazenamento. Quando isto acontece, o ficheiro de *log* que lhe está associado pode ser eliminado. Os dados neste ficheiro são armazenados de forma ordenada, facilitando as pesquisas sobre as chaves;
- Ficheiro de *log*: ficheiro escrito sequencialmente e armazenado de forma persistente.

Este sistema disponibiliza operações que permitem criar cópias de segurança da base de dados mas não oferece replicação. Possui, no entanto, funções para que os utilizadores possam implementar o seu sistema de replicação [31].

Com base neste sistema surgiu um projeto designado RocksDB-Cloud [32] que permite utilizar o serviço de armazenamento S3 para guardar ficheiros, garantido tolerância a falhas e partilha de bases de dados entre diferentes servidores.

2.4 Serviços de Armazenamento na Cloud

2.4.1 Simple Storage Service

O serviço de armazenamento S3 da AWS fornece a possibilidade de alojar e aceder a qualquer quantidade de dados sem restrições temporais e de localização. Permite ter uma solução de armazenamento na *cloud* que garante escalabilidade, fiabilidade e performance [33].

Os conceitos principais deste sistema de armazenamento são os Objetos, entidades fundamentais que são armazenadas no sistema, os *Buckets*, estruturas que guardam os objetos, e as Regiões AWS, que armazenam os *buckets* [33].

No que respeita ao modelo de consistência, a documentação [33] refere que o sistema oferece *"read-after-write consistency"* mas apenas para a escrita de novos objetos no *bucket*. No caso de uma reescrita de um objeto já existente, ou um pedido de um objeto que ainda não exista, apenas é garantido consistência eventual [33].

As classes de armazenamento oferecidas por este serviço são [33]:

- Standard e Reduced_Redundancy: classes de armazenamento para objetos que são acedidos frequentemente;

- Inteligent_Tiering: classe de armazenamento que gere de forma automática os objetos, colocando-os na camada de armazenamento mais adequada, consoante os acessos que são feitos;
- Standard_IA e Onezone_IA: classes de armazenamento adequadas para dados cujo acesso não é frequente;
- Glacier e Deep_Archive: classes de armazenamento para arquivo.

2.4.2 Azure Blob Storage

O Azure Blob Storage [34] é a solução de armazenamento de objetos na *cloud* disponibilizada pela Microsoft.

Os componentes principais deste sistema são a *Storage account*, que fornece um *namespace* único, os *Blobs*, que são os recursos armazenados e os *Containers*, que organizam os *blobs* [34].

As opções de replicação disponíveis são a replicação dentro do mesmo centro de dados, entre centros de dados dentro da mesma região e entre regiões geograficamente separadadas.

As classes de armazenamento disponibilizadas são [34]:

- *Hot*: otimizada para alojar dados que são acedidos frequentemente;
- *Cool*: otimizada para guardar dados que não são acedidos de forma frequente e são guardados, pelo menos, por trinta dias;
- *Archive*: otimizada para alojar dados que raramente são acedidos e são guardados, pelo menos, por cento e oitenta dias.

2.4.3 Google Cloud Storage

O serviço Cloud Storage é a solução de armazenamento na *cloud* oferecida pela Google. De acordo com a documentação é uma solução adequada para armazenar e servir conteúdos para a Web, armazenamento de dados em arquivo ou armazenamento para permitir a recuperação em casos de desastres [35].

Nesta solução, os elementos principais são [35]:

- *Projects*: estrutura à qual pertencem todos os dados. É composta por um conjunto de utilizadores, conjunto de APIs, dados para cobrança, autenticação e monitorização. Um utilizador pode ter um ou vários projetos;
- *Buckets*: estruturas onde os dados são armazenados. Todos os dados colocados na Google Cloud Store são organizados em *buckets*. Podem ser utilizados para organização de dados e controlo dos acessos a esses mesmos dados mas não podem conter outros *buckets*. A sua criação e remoção está limitada pela plataforma.

- *Objects*: referem os objetos que são alojados no serviço. Não há limite no número de elementos que podem ser guardados num *bucket*.
- *Resources*: termo que designa uma entidade na plataforma. São considerados como um recurso - projetos, *buckets* e objetos.

As classes de armazenamento oferecidas neste serviço são [35]:

- Standard: camada de armazenamento adequada para dados que são acedidos frequentemente e/ou alojados por curtos períodos de tempo;
- Nearline: camada de armazenamento adequada para alojamento de dados que são acedidos de forma não frequente. Mais adequada quando se pode tolerar uma menor disponibilidade e quando os dados vão ficar armazenados pelo menos trinta dias;
- Coldline: camada adequada para armazenar dados acedidos de forma não frequente e cujo armazenamento é superior a 90 dias. É uma camada com custos menores que as duas anteriores;
- Archive: camada de armazenamento que oferece os custos mais baixos e adequada para dados cujo armazenamento é superior a 365 dias. Distingue-se pois afirma disponibilizar os dados ao cliente em alguns milisegundos. Apesar dos custos de armazenamento serem baixos, os custos de acesso e de operação são elevados pelo que, de acordo com a fonte, é a camada ideal para dados que são acedidos menos de uma vez por ano.

2.5 Nota Final de Capítulo

Neste capítulo apresentou-se o estado da arte dos serviços *cloud* através da enumeração e descrição de soluções disponibilizadas pelos fornecedores mais influentes no mercado atual. Esta análise permitiu identificar as soluções existentes e que podem ser utilizadas para a construção de sistemas. Foi ainda apresentado um resumo da evolução dos trabalhos realizados na área da CP, permitindo compreender todos os obstáculos superados e identificar os trabalhos mais atuais e quais as suas contribuições. No próximo capítulo será apresentado o resultado do trabalho de investigação realizado com os esquemas de CP mais atuais e que identificou as questões práticas não endereçadas ou deixadas para trabalhos posteriores.

Capítulo 3

Tópicos em Aberto nos Trabalhos sobre Esquemas de Criptografia Pesquisável

Um esquema de Criptografia Pesquisável (CP) por si só não permite realizar operações sobre ficheiros, necessitando de estar integrado num sistema mais complexo para que tal aconteça. A literatura com alguns dos esquemas de CP mais recentes foi analisada para identificar quais os tópicos não endereçados ou deixados para trabalhos posteriores e que devem ser considerados no desenvolvimento de sistemas reais.

Os tópicos deixados em aberto pelos trabalhos académicos de CP e que foram identificados são os seguintes:

- Tópico 1 (T1): Armazenamento e Operações sobre Ficheiros;
- Tópico 2 (T2): Custos Financeiros de Diferentes Soluções e Arquiteturas;
- Tópico 3 (T3): Necessidades da Reindexação e Libertação de Espaço no Índice;
- Tópico 4 (T4): Tratamento dos Nomes Originais dos Ficheiros;
- Tópico 5 (T5): Múltiplas Soluções de Armazenamento.

O resultado da investigação desenvolvida será apresentado neste capítulo, onde cada secção corresponde a um dos tópicos identificados.

3.1 Armazenamento e Operações sobre Ficheiros (T1)

O processamento dos ficheiros antes do armazenamento e o local onde esse armazenamento é feito são fundamentais para que se consiga integrar a manipulação de ficheiros com o esquema de CP e estes tópicos não são endereçados pelos trabalhos académicos. Apenas com essa integração se consegue estender as operações sobre o esquema de CP

para operações sobre ficheiros. Nesta secção identificam-se os aspetos que é necessário considerar no que respeita ao tratamento de ficheiros e às operações que se pretende que o sistema ofereça aos utilizadores.

3.1.1 Onde são guardados os ficheiros?

A solução de armazenamento escolhida condiciona a performance do sistema, os custos de utilização e o tipo de acesso aos dados. Duas soluções possíveis passam por considerar o armazenamento no próprio servidor ou considerar um armazenamento externo, fornecido por outro serviço.

A vantagem de utilizar um serviço de armazenamento externo está na possibilidade de escalar de forma independente em relação ao componente que realiza a computação. No entanto, torna-se importante analisar os custos de acesso, de utilização e regras de armazenamento de ficheiros pois, dependendo da aplicação, o uso de um serviço externo pode ser desvantajoso. De acordo com a análise feita em [9], os serviços de armazenamento (referidos em 2.4) são escaláveis e fornecem aos clientes uma solução de armazenamento a longo prazo a preços acessíveis mas têm a desvantagem de ter custos de acesso elevados e latências altas. Pelo contrário, ao ser utilizado o armazenamento não volátil da própria VM onde está a executar o servidor, as escritas e leituras de ficheiros podem revelar-se mais rápidas e melhorar a performance do sistema mas terá de ser equacionado o custo de acesso a maiores quantidades de memória, caso seja necessário.

3.1.2 Cifra de Ficheiros e Chaves Utilizadas

O tratamento realizado com os ficheiros, para que possam ser alojados de forma segura, é também uma questão deixada em aberto nos trabalhos analisados. Torna-se necessário considerar a forma como os ficheiros vão ser cifrados, as chaves que serão utilizadas para esse fim e de que forma são obtidas e geridas.

Algoritmo e Modo de Cifra Pela rapidez de execução que o sistema deve proporcionar e pelas dimensões consideráveis que os ficheiros podem ter, a solução a adotar deverá utilizar criptografia simétrica. Dependendo do algoritmo utilizado, poder-se-á garantir a autenticidade dos ficheiros obtidos a partir do servidor através de contruções criptográficas *Authenticated Encryption with Associated Data* (AEAD). A escolha da construção criptográfica deverá ter em atenção o número máximo de mensagens que se conseguirá cifrar com uma única chave pois o número de ficheiros armazenados também será pequeno se esse valor for baixo, uma vez que todos os ficheiros são cifrados com a mesma chave.

Geração e Gestão de Chaves Criptográficas A forma como são geradas e geridas as chaves criptográficas, bem como outros elementos necessários à utilização dos algoritmos de cifra dos ficheiros, como é o caso dos *nonces*, não é considerada nos trabalhos estudados. Outra questão importante prende-se com a necessidade de trocar as chaves criptográficas

utilizadas. Se forem trocadas as chaves criptográficas que são utilizadas para operar sobre o índice, é necessário executar uma operação de reindexação, que se resume à troca de todas as entradas do índice por novas entradas obtidas com as novas chaves. Se forem também trocadas as chaves de cifra dos ficheiros, todos os ficheiros terão de ser acedidos no cliente e cifrados com uma nova chave.

3.1.3 Inserção e Pesquisa de Ficheiros

A forma como são realizadas as inserções e pesquisas sobre o índice é apresentada e descrita nos documentos dos esquemas de CP em detalhe mas a real manipulação de ficheiros e a interligação desse tratamento no esquema é deixada em aberto.

Ao introduzir a manipulação de ficheiros reais pode revelar-se mais informação ao fornecedor de serviços. Se for inserido apenas um documento acompanhado da colocação das entradas no índice, é possível relacionar as entradas com o ficheiro inserido. A inserção de ficheiros necessita ainda de um tratamento prévio de obtenção das palavras-chave. Esta extração está dependente do tipo de ficheiro que se pretende inserir e do tipo de pesquisas que se pretende efetuar. Quanto mais palavras-chave forem obtidas do ficheiro inserido, maiores serão as possibilidades de uma pesquisa retornar o documento. Por outro lado, se forem extraídas muitas palavras-chave, o mapa de contadores e o índice terão dimensões maiores e consequentemente o sistema utilizará mais espaço de armazenamento e realizará maior número de computações sobre estas estruturas de dados. Desta forma, quanto mais apurado for o elemento do sistema que extrai as palavras dos documentos, mais eficiente será a inserção e a realização das pesquisas pois existirão menos entradas no índice mas, as existentes, descrevem de maneira adequada o documento.

3.1.4 Remoção de Ficheiros do Armazenamento

A remoção de ficheiros de memória não volátil e a forma como se ligam às operações realizadas sobre o índice é também uma questão que não é abordada nos trabalhos estudados. Para que o utilizador do sistema saiba que documento pretende remover, tem de realizar uma operação de pesquisa para o identificar e, após confirmação, o ficheiro tem de ser analisado para determinar todas as palavras-chave que contém, pois só desta forma todas as entradas no índice podem ser removidas. Para todas as palavras-chave encontradas, são enviadas entradas para o índice de remoções, uma vez que se tal não acontecer vão existir entradas no índice de inserções que não pertencem a nenhum ficheiro armazenado. À semelhança do que acontece com a operação de inserção, se apenas um documento for removido, é possível associar as entradas das remoções lógicas com o ficheiro indicado para eliminação.

A operação de remoção de ficheiros provoca o aumento do espaço de armazenamento utilizado pelo índice. A recuperação deste espaço é possível através da eliminação física das entradas no índice e existem duas possibilidades de o fazer, dependendo do esquema de CP considerado. Nos casos em que o espaço ocupado pelas remoções lógicas no índice

só é reclamado através de uma operação de reindexação, não é necessário que o cliente execute mais nenhuma ação após a indicação ao servidor para eliminar o ficheiro do armazenamento não volátil. No caso em que o armazenamento ocupado pelo índice é reclamado na sequência de pesquisas ou de reindexação, após solicitar ao servidor a eliminação do ficheiro em questão, o cliente tem duas alternativas: ou executa pesquisas com as palavras-chave apagadas para que estas possam ser eliminadas ou aguarda pela operação de reindexação.

3.1.5 Edição de Ficheiros

Num sistema que suporte a manipulação de ficheiros faz sentido que possibilite operações de edição, em que o utilizador altera os ficheiros, inserindo e retirando palavras. A possibilidade de remoção de entradas de forma lógica no índice dá suporte a esta funcionalidade mas é necessário avaliar a forma como pode ser integrada com a manipulação do ficheiro. Tal como na remoção de ficheiros, a edição de um ficheiro ocorre após uma operação de pesquisa que devolve o ficheiro ao utilizador.

A edição de ficheiros pode realizar-se utilizando duas abordagens diferentes. A primeira abordagem caracteriza-se por não remover o ficheiro do sistema antes de iniciar a edição e a segunda abordagem remove o ficheiro por completo do sistema e, quando terminada a edição, o ficheiro é tratado como se fosse uma nova inserção.

Na primeira abordagem, ao terminar a edição do ficheiro, vão existir entradas que correspondem a novas inserções no índice (se o utilizador adicionou novas palavras), remoções lógicas no índice (se o utilizador apagou palavras no documento obtido pela pesquisa) ou palavras que se mantêm no documento (e por isso não são inserções nem remoções). Esta situação implica que o cliente tenha de ter presente a informação contida no documento antes da edição iniciar, permitindo no final categorizar as palavras como inserção, remoção ou sem alteração. Esta abordagem não permite que seja trocado o nome do ficheiro após a edição, uma vez que as entradas que já estavam no índice e que não foram alteradas mapeiam a chave para esse valor.

A segunda abordagem passa por realizar uma operação de remoção completa do ficheiro antes de iniciar a operação de edição. Esta alternativa permite que, após o fim da edição, todas as palavras que forem selecionadas como "palavra-chave" são inserções no índice e que seja trocado o nome cifrado do ficheiro editado, pois todas as entradas que correspondiam ao nome do ficheiro antes de ser editado já foram removidas de forma lógica.

No entanto, um problema comum a estas duas abordagens reside no suporte de "edições consecutivas" de ficheiros, que provoca um aumento de entradas no índice. Vão gerar-se entradas repetidas, onde diferentes chaves da mesma palavra mapeiam para o mesmo nome de ficheiro. Esta situação torna necessário adaptar o algoritmo de pesquisa sobre os índices, que terá de fazer a diferença entre o total de inserções da palavra no ficheiro e o total de remoções.

3.2 Custos Financeiros de Diferentes Soluções e Arquiteturas (T2)

Outra das questões não referidas na literatura sobre os esquemas de CP é o impacto que a arquitetura e as soluções tecnológicas escolhidas podem ter nos custos imputados ao cliente.

Analisando as soluções propostas pelos fornecedores de serviços na *cloud* e os preços cobrados, verifica-se que é necessário considerar a quantidade de armazenamento, o serviço de alojamento que será utilizado e que o custo da computação pode ser uma percentagem considerável dos custos totais, dependendo do tempo de funcionamento e capacidade da VM. Os trabalhos analisados assumem a existência de computação num servidor na *cloud* mas explorar soluções diferentes poderá ser interessante, pelo que se justifica analisar outras arquiteturas de sistema que utilizem apenas a computação no cliente e os respetivos modelos de custos.

3.3 Necessidades da Reindexação e Libertação de Espaço no Índice (T3)

As operações de reindexação de um esquema de CP permitem atingir três objetivos. O primeiro permite que sejam libertadas entradas do índice, reduzindo o espaço de armazenamento ocupado no servidor. O segundo objetivo é permitir a troca de chaves criptográficas utilizadas e, por fim, esconder padrões de acesso ao índice. Quando o esquema de CP é utilizado num sistema em que existe uma real manipulação de ficheiros, as operações de reindexação devem também permitir a troca de chaves criptográficas utilizadas na cifra dos ficheiros armazenados.

Ao remover e editar ficheiros são colocadas entradas nos índices que indicam que determinada palavra foi retirada de um documento e estas provocam o aumento do espaço de armazenamento ocupado. Essas entradas designam-se por "remoções lógicas" [36]. Nesta secção analisaremos a operação de reindexação, que permite reduzir o espaço de armazenamento utilizado pelo índice através da eliminação física das entradas de remoções lógicas.

Os esquemas estudados realizam as remoções lógicas através da inserção de novas entradas no índice mas apenas alguns permitem a sua eliminação física. O esquema genérico B'(Σ) (quando instanciado, origina os esquemas Moneta e Fides [20]) e o esquema Mitra [15] já prevêem a remoção física de entradas do índice, sendo estas operações realizadas durante as pesquisas. O servidor, ao responder a uma pesquisa, remove fisicamente as entradas e liberta espaço de armazenamento, como ilustrado na figura I.8 (linha 2) e na figura I.7 (linhas 8 a 11). O cliente filtra a resposta recebida, eliminando as entradas que correspondem a *deletes*, volta a cifrar as restantes com uma nova chave e envia para o servidor [15, 20]. Esta construção do esquema permite que a libertação de espaço possa

ser realizada sempre que se elimina ou edita um ficheiro, pesquisando as palavras removidas na operação. Aproveitando esta construção, a reindexação pode ser implementada pesquisando todas as palavras presentes no mapa de contadores pois, no final, apenas as entradas que correspondem a inserções estarão presente no índice.

Os restantes esquemas estudados não fazem a remoção física de entradas na sequência de pesquisas. Os esquemas sophos [2] e $\text{Diana}_{\text{del}}$ [20] realizam as remoções da mesma forma, mantendo duas instâncias das estruturas de dados, sendo uma utilizada para as operações de inserção e outra para as operações de remoção [20]. O esquema Janus utiliza duas instâncias de um esquema dinâmico de CSP que seja *forward private* e que apenas suporte inserções, sendo as remoções tratadas por uma das instâncias [20]. Para estes esquemas, a reindexação que pesquise todas as palavras presentes no mapa de contadores será realizada através da criação de um novo índice inserções, que será preenchido com os resultados obtidos da pesquisa sobre o antigo índice.

Todos os esquemas referidos permitem ainda a implementação da operação de reindexação através de um novo processamento dos ficheiros armazenados. O cliente solicita ao servidor que devolva todos os ficheiros que tem armazenados e estes são novamente processados para extração de palavras-chave. Esta solução não obriga a utilizar em simultâneo dois mapas de contadores e dois índices, bastando eliminar os antigos e criar novos para que se possa iniciar o tratamento dos ficheiros. Esta opção é também adequada para os esquemas Orion e Horus, que apesar de tratarem as remoções de forma lógica, apresentam algumas diferenças em relação aos esquemas já analisados. O esquema Orion [15] lida com as remoções das entradas do índice através da inserção de entradas específicas nos *Oblivious Maps*, ilustrado nas linhas 15 a 33, na figura I.9. Quando existe uma remoção, a entrada do par *(w, id)* no mapa $OMAP_{upd}$ é colocada com o valor -1, o que indica que as entradas dos pares apagados continuam a ocupar espaço em memória. O esquema Horus funciona de forma semelhante ao Orion, mas substitui o $OMAP_{src}$ por uma estrutura *Path-ORAM* não recursiva. Estes esquemas, devido às estruturas de dados que utilizam, apresentam a desvantagem de ter de alocar o espaço de armazenamento necessário no servidor para a quantidade total de atualizações que se prevê que sejam feitas. Esta característica limita a utilização num sistema real mas é proposta uma alternativa, que consiste a cada 2^i atualizações fazer o *download* das estruturas de dados do servidor e reinicializá-los com mais um nível $(i+1)$, fazendo uso de novas chaves criptográficas [15]. Em virtude desta característica, a operação de reindexação pode ser feita fazendo o *download* dos ficheiros e o seu processamento em vez de solicitar o envio das estruturas de dados do servidor para o cliente. Desta forma podem trocar-se as chaves criptográficas utilizadas para cifrar os ficheiros, as chaves criptográficas do esquema de CSP e instanciar as novas estruturas de dados com maior capacidade para as operações de *update*.

3.4 Tratamento dos Nomes Originais dos Ficheiros (T4)

Os nomes dos ficheiros e as extensões revelam informações sobre o seu conteúdo. Nos vários trabalhos analisados, os nomes dos ficheiros são referidos como identificadores ou *index* e é através deles que os ficheiros são encontrados na memória não volátil, onde estão armazenados. Constatou-se que a questão da transformação dos nomes originais em nomes de armazenamento não é referida.

O estudo realizado permitiu dividir os vários esquemas em dois grupos, de acordo com o componente que decifra o nome de armazenamento do ficheiro. O primeiro grupo contém os esquemas onde o servidor, pela informação que dispõe, efetua a decifra dos nomes de armazenamento dos ficheiros. O segundo grupo engloba os esquemas onde os nomes dos ficheiros são decifrados no cliente e posteriormente enviados, em claro, para o servidor.

3.4.1 Servidor Decifra Nomes dos Ficheiros

No esquema Sophos [2] é o servidor que decifra e acede diretamente ao *index* ou nome de armazenamento. Como se pode ver na figura I.1, aquando das operações de inserção de documentos na linha 10, o nome de armazenamento do ficheiro é cifrado (representado pela letra *e*). No entanto, quando é realizada uma operação de pesquisa, o nome cifrado *e* é obtido pelo servidor na linha 8 mas é ainda obtido o nome pelo qual o ficheiro estará armazenado, representado por *ind*, na linha 9. O servidor, em virtude da informação que tem de conhecer para realizar as pesquisas (chave gerada para a palavra-chave pesquisada K_w e o *Search Token* (ST)), fica a conhecer o nome de armazenamento do ficheiro.

A mesma questão surge no pseudo-código apresentado para esquemas de CSP com *forward privacy* em [20], ilustrado na figura I.2. Na linha 9 da operação de pesquisa surge o nome de armazenamento do documento em claro, no servidor. Também o esquema de CP Janus [20], por forma a permitir que o servidor consiga devolver os resultados ao cliente, acede aos nomes de armazenamento dos documentos. A figura I.3 ilustra esse acesso, na linha 8 da operação de pesquisa. Outro esquema que também apresenta a mesma característica é o esquema de Etemad et al. [37], como ilustrado na figura I.4.

Estes esquemas obrigam a que quando é introduzido um ficheiro no sistema, o nome de armazenamento ou *index* seja já uma transformação do nome original. Esta transformação tem de ser compatível com o sistema operativo, evitando caracteres proibidos, e compatível com as operações criptográficas que serão realizadas, pelo que o seu tamanho também estará condicionado.

3.4.2 Envio dos Identificadores em Claro para o Servidor

No esquema genérico B'(Σ) [20] é o cliente que decifra os elementos que permitem conhecer os nomes de armazenamento obtidos na sequência de uma pesquisa mas, para obter os ficheiros, tem de devolver esses nomes ao servidor em claro. Esta questão é

ilustrada na figura I.8 onde se pode verificar que, apesar dos pares que contêm o nome do ficheiro e a operação realizada chegarem cifrados ao cliente na linha 5, após serem filtradas as entradas que não foram eliminadas, o cliente solicita os ficheiros através do nome *ind*. O esquema de CSP Mitra [15] executa o tratamento dos identificadores dos ficheiros de forma igual. O servidor não lhes consegue aceder pois não dispõe da informação necessária, sendo estes apenas decifrados no cliente, na sequência de uma pesquisa. A decifra é realizada nas linhas 13 a 15 do algoritmo de pesquisa do esquema referido, ilustrado na figura I.7.

O esquema de CSP Orion [15] envia também os identificadores dos ficheiros em claro, quer nas operações de pesquisa, quer nas operações de atualização, sejam elas de inserção ou remoção, como ilustrado pelas linhas 1 e 2 da figura I.9 e pela figura I.10. Uma versão modificada deste esquema, denominada de Horus [15], apresenta a mesma propriedade no algoritmo de atualização e pesquisa.

3.5 Multiplas Soluções de Armazenamento (T5)

Nenhum dos trabalhos analisados prevê a utilização de serviços de armazenamento de diferentes fornecedores para o mesmo índice. A possibilidade de utilizar vários serviços de armazenamento dá flexibilidade ao utilizador de mudar o local de armazenamento mantendo o sistema em funcionamento ou ter a sua base de dados divida por serviços de armazenamento de fornecedores diferentes.

3.6 Tópicos Selecionados para Implementação

Dos tópicos apresentados apenas alguns foram selecionados para implementação. O trabalho desenvolvido na sequência desta investigação estudou e implementou todas as operações sobre ficheiros (com exceção da edição), a operação de reindexação e o tratamento dos nomes originais dos ficheiros. O tópico do suporte para multiplas soluções de armazenamento não foi endereçado nas implementações realizadas.

3.7 Nota Final de Capítulo

Ao longo deste capítulo foram apresentados os resultados da investigação realizada aos esquemas de CP mais atuais. Os tópicos enunciados, apesar de não estarem diretamente relacionados com o funcionamento dos esquemas, são imprescindíveis para que estes possam efetivamente operar sobre ficheiros, ao invés de fazer apenas pesquisas no índice.

Se alguns dos aspetos referidos possuem soluções possíveis de analisar do ponto de vista teórico, como os algoritmos de cifra que podem ser utilizados ou os custos de utilização, outros mais práticos necessitam de um sistema real para que as soluções possam ser testadas, como é o caso da performance. Verificou-se que seria necessário testar mais que

uma arquitetura de sistema para perceber que soluções seriam mais adequadas, de acordo com a oferta atual de serviços existentes na *cloud*. Desta forma, no próximo capítulo são apresentadas as arquiteturas desenvolvidas e será feita uma descrição mais detalhada do esquema de CP que servirá de base à implementação das arquiteturas.

Capítulo 4

Modelos e Bases

Uma vez identificadas as questões não endereçadas pelos trabalhos mais recentes na área da CP, que resultaram da investigação realizada e que foram apresentadas no capítulo 3, é necessário desenvolver arquiteturas que permitam aprofundar o seu estudo. Neste capítulo é apresentado o modelo de adversário que foi considerado, conceitos, o esquema de CP utilizado como base e as arquiteturas desenvolvidas.

4.1 Modelo de Adversário

O modelo de adversário considerado é aquele em que um servidor não se desvia do protocolo implementado mas tenta aprender o máximo sobre os documentos do cliente [19]. Assim, serão considerados adversários passivos e que atuam de forma *honest-but-curious* [2]. São exemplo deste tipo de adversário o fornecedor de serviços de *cloud*, que consegue sempre aceder aos conteúdos que são colocados nos seus servidores.

4.2 Conceitos

Torna-se importante clarificar os conceitos que serão utilizados ao longo da descrição das arquiteturas e da apresentação dos sistemas desenvolvidos.

Cliente O cliente representa qualquer aparelho com possibilidade de se ligar ao sistema, com capacidade de realizar as computações necessárias e espaço de armazenamento para as estruturas de dados. O utilizador interage com o sistema através da máquina cliente.

Servidor O servidor é o componente com o qual o cliente comunica. Tem capacidade de realizar computações e é responsável pelo armazenamento dos ficheiros. Para conseguir

realizar esta tarefa tem de executar operações sobre o índice e sobre a base de dados, independentemente do local onde estes dois elementos se encontrem fisicamente.

Mapa de Contadores Designa a estrutura de dados do cliente que permite encontrar o contador que está associado a cada palavra-chave. O valor do contador representa o número de documentos no sistema que contêm a palavra-chave. Por forma a suportar remoções, esta estrutura tem de ser duplicada, permitindo guardar os contadores de remoções que representam o número de documentos de onde a palavra-chave foi eliminada.

Índice Designa as estruturas de dados utilizadas pelo servidor e que permitem encontrar os nomes cifrados dos documentos solicitados pelo cliente, na sequência de uma pesquisa. À semelhança do que acontece com os mapas de contadores, para suportar remoções o índice tem de ser duplicado.

Base de Dados Designa o conjunto de documentos cifrados que estão armazenados em memória não volátil. É operada pelo servidor.

Remoção Lógica Entrada colocada no índice de remoções que representa a remoção de um par (palavra-chave, documento) do sistema. Na realização de uma operação de pesquisa, o servidor procura no índice de remoções se existe uma entrada que corresponda à entrada encontrada no índice de inserções. Caso se verifique, significa que a palavra pesquisada foi removida do documento encontrado.

Nome de Armazenamento Designa-se por nome de armazenamento, ou *index*, o nome que é atribuído ao ficheiro após a sua cifra, pelo cliente. Este *index* substitui assim o nome original do ficheiro e apenas o cliente consegue fazer a conversão de nome original para *index* e vice-versa.

Nome Cifrado Designa a transformação criptográfica do nome de armazenamento, e que corresponde ao valor colocado nas entradas do índice.

4.3 Esquema Base de Criptografia Pesquisável

O esquema de criptografia pesquisável escolhido para incorporar o sistema designa-se por *Σοφος* ou Sophos [2]. A escolha ficou a dever-se ao facto deste esquema ser recente, fazer parte do estado da arte, estar implementado e disponível de forma livre [38] e ainda por garantir *forward privacy*, resistindo assim aos ataques adaptativos analisados nos trabalhos de [6] e [3]. Não consegue, no entanto, inutilizar os ataques não adaptativos [2].

Este esquema, na solução original proposta pelo autor [2, 38], faz uso de duas estruturas de dados, uma localizada no dispositivo cliente e designada por $\boldsymbol{W}$ (mapa de

contadores) e uma outra localizada no servidor designada por $\boldsymbol{T}$ (índice). Para o funcionamento do esquema são ainda utilizados *tokens*, respetivamente *Search Tokens* (STs) e *Update Tokens* (UTs). Quando o cliente pretende pesquisar os documentos que contêm uma dada palavra-chave w, verifica as entradas no mapa de contadores $\boldsymbol{W}$ e obtém o contador associado c, que permite calcular o ST. De uma forma mais concreta, calcula $ST_c(w)$ a partir do $ST_0(w)$ e utilizando o valor de c guardado em $\boldsymbol{W}$, operação representada pela seta vermelha na figura 4.1. O $ST_c(w)$ é posteriormente enviado para o servidor para que este possa procurar os identificadores dos ficheiros pretendidos [2]. Os UTs correspondem à localização lógica do nome cifrado do documento no índice no servidor.

A pesquisa efetuada pelo servidor sobre o índice é realizada no sentido contrário, permitindo encontrar todos os documentos onde a palavra-chave pesquisada se encontra. Esta pesquisa está ilustrada pelas setas pretas na figura 4.1. O servidor faz uso da chave pública do cliente (*Public Key* (PK)) para iterar por todos os ST já gerados para a palavra procurada, obtendo o correspondente UT e, a partir deste, acede ao identificador cifrado do documento. [2].

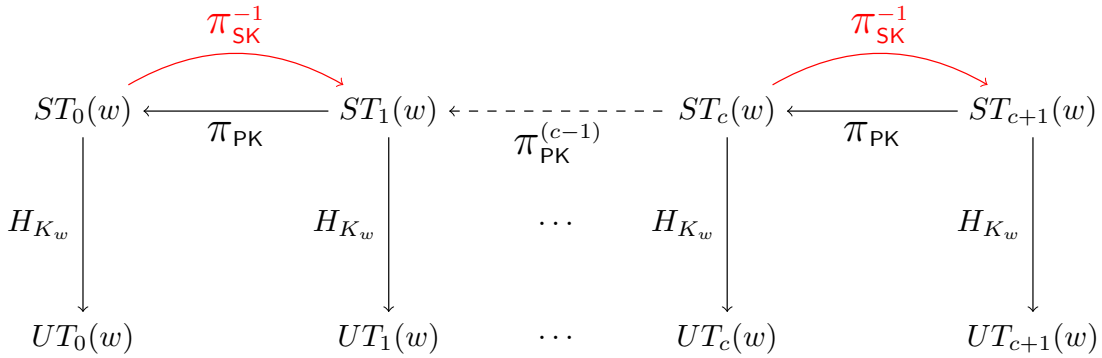


Figura 4.1: Relação entre *Search Tokens* (STs) e *Update Tokens* (UTs) [2]

Para que este esquema suporte remoções, é necessário duplicar o mapa de contadores e o índice (ou adaptar o índice para que passe a utilizar um marcador de remoções associado ao identificador do documento). Passa a existir o índice de inserções que recebe as entradas que correspondem a inserções de palavras em documentos e o índice de remoções que recebe as entradas com as remoções lógicas. Com esta alteração, sempre que uma palavra for pesquisada, o servidor terá de verificar os valores obtidos dos dois índices para determinar se a inserção foi posteriormente removida [2]. Para evitar que os pares chave-valor colocados no índice de remoções sejam iguais aos colocados no índice de inserções quando é realizada uma operação de remoção, é ainda necessário duplicar as chaves criptográficas utilizadas pelo esquema.

4.4 Arquiteturas

Nesta secção serão descritas as duas arquiteturas propostas para implementação e que permitem estudar os tópicos que resultaram do trabalho de investigação realizado aos esquemas mais recentes de CP, apresentado no capítulo 3.

4.4.1 Solução com Computação com Armazenamento

A arquitetura apresentada e ilustrada na figura 4.2 é uma arquitetura cliente-servidor com um cliente único e um servidor a funcionar numa VM, num serviço de *cloud*. Esta é a arquitetura base do Sophos, tendo sido modificada para suportar as operações sobre ficheiros e reindexação. O cliente é responsável por alojar os dois mapas de contadores necessários ao funcionamento do esquema de CP e o servidor por alojar os dois índices e a base de dados.

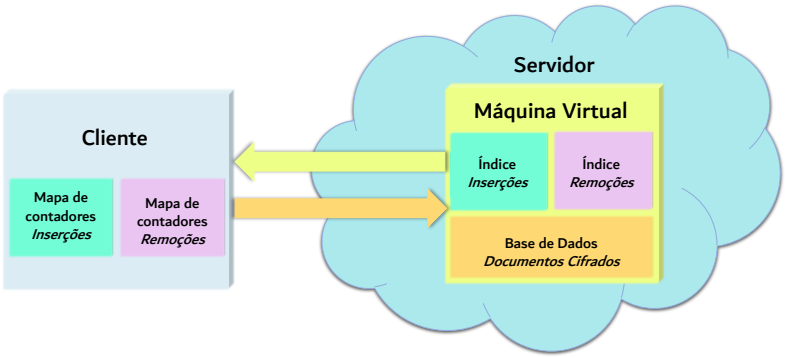


Figura 4.2: Diagrama ilustrativo da arquitetura "computação com armazenamento".

4.4.2 Solução Apenas com Armazenamento

Importa também analisar soluções que possam ser utilizadas por clientes que possuam mais capacidade de computação. A arquitetura é composta pelo cliente, que executa todas as computações do esquema de CP na sua própria máquina, e o serviço de armazenamento na *cloud* que aloja o índice e a base de dados, como ilustrado na figura 4.3.

Esta arquitetura pode fazer uso de uma *cache* que permitirá reduzir o número de pedidos ao serviço de alojamento de ficheiros, e de um *buffer* para agrupar entradas do índice antes do seu envio para o serviço de armazenamento, como ilustrado na figura 4.4. O *buffer* irá conter as entradas mais recentes do índice de inserções e do índice de remoções.

4.5 Notas Finais de Capítulo

Neste capítulo foram apresentadas as bases, os conceitos, o modelo de adversário e as arquiteturas que materializam o ponto de partida para as implementações realizadas.

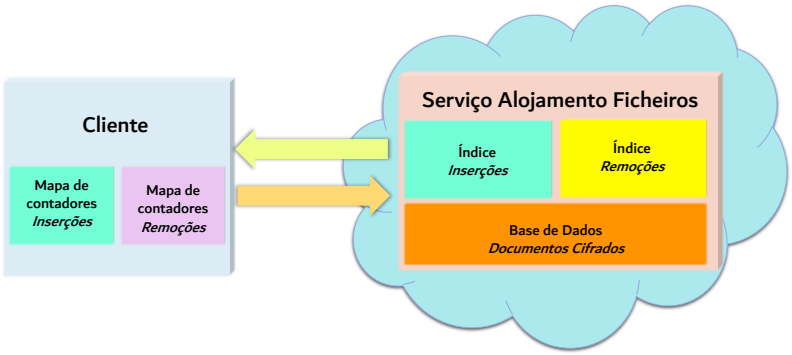


Figura 4.3: Diagrama ilustrativo da arquitetura "apenas armazenamento". Toda a computação é realizada pela máquina cliente.

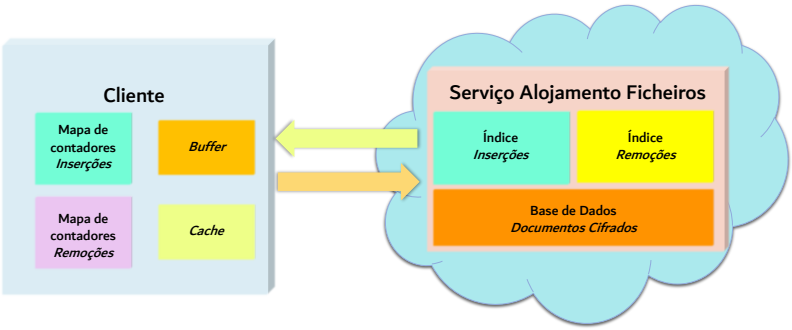


Figura 4.4: Diagrama ilustrativo da arquitetura "apenas armazenamento" com utilização de *buffer* e *cache* no cliente.

O próximo capítulo descreve a implementação do Sophos que foi utilizada como base e apresenta as alterações realizadas, que resultaram nos três sistemas desenvolvidos.

Capítulo 5

Implementação

A implementação das duas arquiteturas teve como objetivo aprofundar o estudo de alguns dos tópicos identificados pelo trabalho de investigação apresentado no capítulo 3. Neste capítulo serão descritas as três implementações realizadas, desenvolvidas a partir do código do Sophos disponível em [38]. Para distinguir o código base daquele que foi escrito no âmbito da dissertação, apresenta-se na primeira secção o código do Sophos que foi utilizado e nas três secções seguintes são descritas as novas classes criadas e métodos adicionados a cada sistema. No resto do capítulo é detalhada a implementação das operações sobre ficheiros, das reindexações e da forma de tratamento dos nomes.

5.1 Implementação de Base

O esquema Sophos foi o esquema de CP escolhido e está escrito em C/C++ [38]. A implementação disponível permite realizar operações de inserção e pesquisas no índice. O sistema inicia a sua execução carregando um ficheiro JSON (que mapeia palavras em listas de números inteiros) para o mapa de contadores do cliente e para o índice no servidor. Após esta operação, o utilizador pode realizar pesquisas no sistema, obtendo como resposta os números inteiros no terminal do cliente.

As classes e ficheiros disponibilizados pela implementação do Sophos que foram utilizados como ponto de partida para o desenvolvimento dos sistemas apresentados são:

- src/sophos_client.cpp: ficheiro que contém a função `main()` do cliente e implementa as funcionalidades que permitem ao utilizador indicar o ficheiro JSON para a inserção de entradas no índice e realizar as pesquisas;

- sophos_client_runner.h: ficheiro que declara a classe SophosClientRunner, que permite comunicar com o servidor e implementa os métodos que criam as mensagens utilizadas pelo gRPC [39]. A implementação está no ficheiro `sophos_client_runner.cpp`;
- sophos_client.hpp: ficheiro que declara a classe SophosClient, implementada no ficheiro `sophos_client.cpp`. Esta classe implementa os métodos que interagem diretamente com os mapas de contadores do cliente;
- src/sophos_server.cpp: ficheiro que implementa o código de criação do gRPC Server e inicializa a biblioteca criptográfica *libsodium* [40].
- sophos_server_runner.hpp: declara as classes `SophosServerRunner` e `SophosImpl`. A primeira é responsável por realizar as operações necessárias para colocar em funcionamento o servidor gRPC e inicializar o objeto da classe `SophosImpl`. A classe `SophosImpl` está implementada no ficheiro `sophos_server_runner.cpp` e permite realizar as operações sobre o índice cifrado no servidor;
- sophos_server.hpp: ficheiro que declara a classe `SophosServer`, responsável por comunicar com a classe `RockDBWrapper` e obter as informações necessárias do índice. Esta classe está implementada no ficheiro `sophos_server.cpp`;
- sophos_common.hpp: declara um método criptográfico utilizado no esquema de CP e que é utilizado tanto pelo cliente como pelo servidor e as `struct` de apoio à construção das mensagens gRPC trocadas entre o cliente e o servidor. A implementação do método é feita no ficheiro `sophos_common.cpp`;
- rocksdb_wrapper.hpp: ficheiro que implementa a classe `RockDBWrapper`, que utiliza o SGBD RocksDB como índice no servidor e declara a classe `RockDBCounter` que utiliza o SGBD RocksDB como mapa de contadores no cliente. A implementação da classe `RockDBCounter` está no ficheiro `rocksdb_wrapper.cpp`;
- logger.hpp: declara as classes que permitem fazer leituras de performance da implementação do Sophos. A implementação está no ficheiro `logger.cpp`.;
- thread_pool.hpp: implementa uma *pool* de *threads*.
- utils.hpp: implementa métodos que permitem aos esquema de CP interagir com o sistema de ficheiros, realizar conversões para formato hexadecimal e operações de *exclusive-or* com inteiros de 64 *bits*. A implementação está feita no ficheiro `utils.cpp`.
- Diretoria protos: contém os ficheiros que permitem definir o serviço gRPC.

Os componentes cliente e servidor da implementação do esquema de CP comunicam recorrendo à *framework* gRPC [39], que utiliza os *protocol buffers* como *Interface Definition Language* (IDL) [41]. Os métodos já implementados eram:

- setup(): que permite ao cliente configurar a sua ligação ao servidor na primeira ligação estabelecida;
- search(): que permite enviar um pedido de pesquisa para a palavra-chave indicada pelo utilizador;
- insert(): método que permite introduzir uma nova entrada no índice;
- bulk_insert(): método que permite enviar entradas para o índice através de uma sequência de mensagens. O servidor recebe todas as mensagens e envia a resposta ao cliente.

A implementação do Sophos utiliza o SGBD RocksDB para implementar o mapa de contadores no cliente e o índice no servidor [2, 30, 31].

No que respeita às bibliotecas criptográficas utilizadas para suportar as operações descritas, é utilizado o algoritmo RSA da biblioteca *OpenSSL BigNum* para as *trapdoor permutations* [2]. A função pseudoaleatória F e as funções H, descritas no mesmo documento e cuja utilização está ilustrada na figura I.1, são instanciadas como *Hashed Message Authentication Code* (HMAC), com a função de *hash* Blake2b [2, 42].

5.1.1 Paralelização

A implementação do Sophos recorre à paralelização nas operações de inserção de entradas no sistema e na realização de pesquisas no índice.

Nas operações de inserção, as entradas presentes no ficheiro JSON e que mapeiam palavras-chave em conjuntos de números inteiros, são divididas por uma *pool* de *threads* que enviam os pares (*Update Token* (UT), nome cifrado) para o servidor.

Na realização de operações de pesquisa sobre o índice, sempre que o contador associado à palavra procurada é superior a 2 (dois), a operação é realizada em paralelo no servidor. Cada *thread* calcula o *Search Token* (ST) que lhe corresponde, dado pela posição i que ocupa no vetor de *threads* na primeira execução e $i + N$ nas seguintes (com N o número total de *threads* e $i + N < contador$). Uma vez calculado o ST obtém-se o UT e o nome cifrado do ficheiro a ele associado. A figura 5.1 ilustra a forma como as operações são divididas pelas *threads*.

5.2 Implementação da Arquitetura "Computação com Armazenamento"

Nesta secção são apresentadas as novas classes criadas e as extensões adicionadas à implementação do Sophos e que materializam o sistema desenvolvido. O sistema será referido como Sistema Computação com Armazenamento (SCA) no resto do documento.

O trabalho realizado integra as operações sobre ficheiros de texto (extensão `.txt`) com o esquema Sophos. Foi necessário adaptar os métodos de inserção e pesquisa existentes

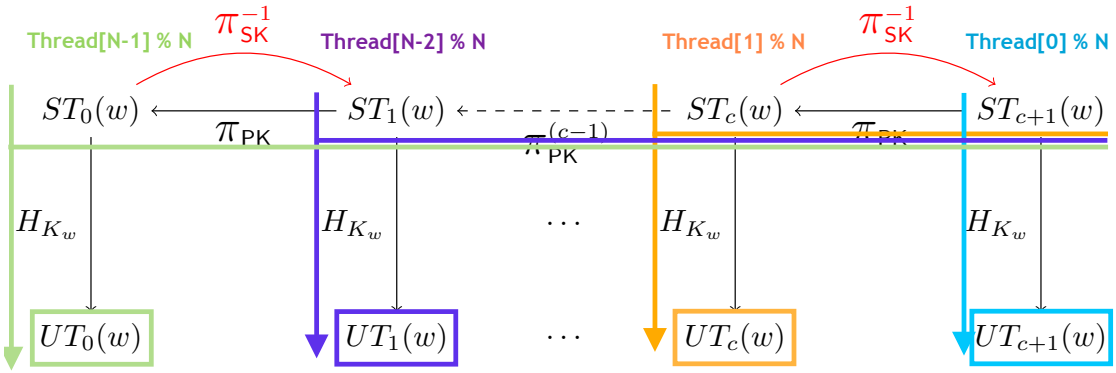


Figura 5.1: Ilustração da paralelização das pesquisas no índice, na implementação de base do esquema Sophos.

para lidarem com ficheiros e desenvolver toda a estrutura de suporte de remoções e reindexações. As operações disponíveis no sistema são:

1. **Pesquisa:** em que o cliente fornece a palavra-chave (ou palavras-chave) que pretende pesquisar e recebe os documentos que contêm a palavra encontrados na base de dados;

2. **Inserção de documentos:** O utilizador indica o(s) ficheiro(s) que pretende que seja(m) inserido(s) no sistema. Esta inserção pode realizar-se através da indicação do nome do ficheiro a inserir ou selecionando a opção de inserir todos os ficheiros que estiverem na diretoria `inputFiles`;

3. **Eliminação de documentos:** na sequência de uma pesquisa, o utilizador pode remover documentos do sistema. Podem ser removidos todos os documentos obtidos na pesquisa ou selecionar o documento a eliminar pelo nome do ficheiro;

4. **Reindexação parcial:** o cliente pode fazer uma reindexação parcial que consiste na geração de novas chaves criptográficas para um novo índice. Este índice é preenchido apenas com as entradas dos ficheiros que não foram eliminados nas operações anteriores. Esta operação liberta espaço de armazenamento ocupado pelo índice de remoções e troca as chaves criptográficas utilizadas pelo esquema de CP;

5. **Reindexação total (1):** efetua todas as operações realizadas pela reindexação parcial e cifra os documentos armazenados com uma nova chave criptográfica simétrica. Esta operação permite libertar o espaço de armazenamento no índice, ocupado pelas remoções lógicas, e trocar todas as chaves criptográficas no sistema;

6. **Reindexação total (2):** efetua uma reindexação total com um algoritmo diferente, evitando a realização de pesquisas sobre o índice. Obtém os dados através de um novo processamento dos ficheiros armazenados.

Nos restantes pontos desta secção são apresentadas as novas classes adicionadas ao sistema e as adaptações feitas às classes já existentes no código utilizado como base. A figura 5.2 apresenta um diagrama simplificado da interação entre os componentes.

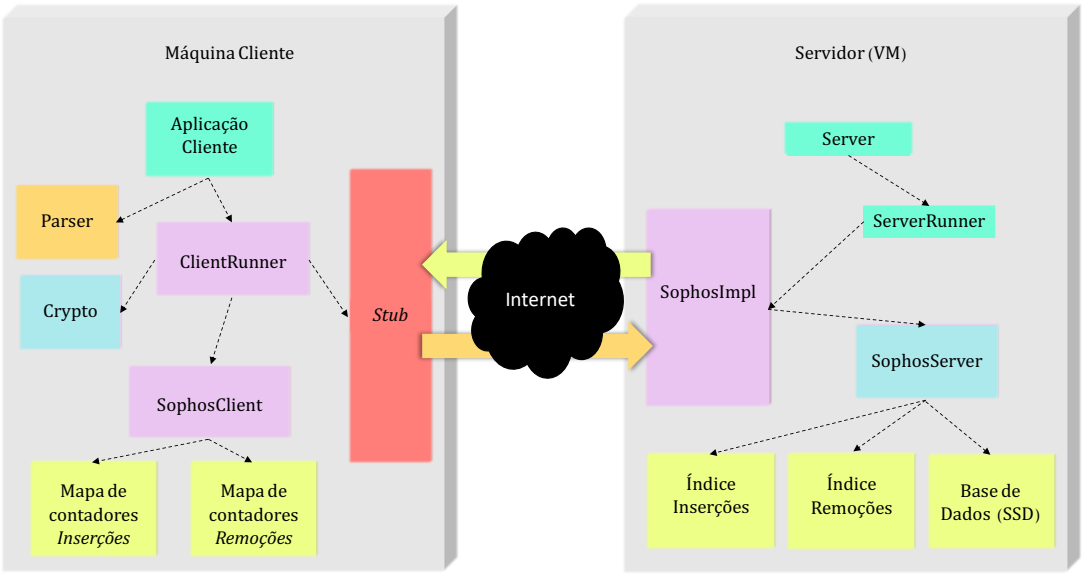


Figura 5.2: Diagrama simplificado do Sistema Computação com Armazenamento (SCA).

5.2.1 Aplicação Cliente

A função `main()` no ficheiro `src/sophos_client.cpp` foi substituída por um cliente interactivo que simula uma aplicação executada em linha de comandos. Este cliente está implementado no ficheiro `src/sysAppClient/interactive_client.cpp`.

A aplicação cliente `interactive_client` começa por fazer a inicialização das bibliotecas criptográficas necessárias ao funcionamento do sistema, nomeadamente *libsodium* [40] e *relic* [43] e acede às configurações do utilizador através de um ficheiro de propriedades. Cria um canal gRPC com o endereço do servidor e inicializa um objeto `SophosClientRunner`. É ainda inicializado um objeto `FileParser`, responsável por extraír as palavras-chave dos documentos e um objeto `Crypto`, que implementa as operações criptográficas realizadas sobre os ficheiros.

5.2.2 Classe FileParser

Esta classe instancia o objeto que é responsável por analisar os ficheiros que o cliente pretende inserir no sistema e por controlar os nomes de armazenamento que são atribuídos. A implementação desenvolvida analisa ficheiros com extensão `.txt` e extrai as palavras-chave do documento. Estas palavras-chave são inseridas numa estrutura de dados auxiliar designada por `invertedIndex` e que mapeia as palavras-chave nos nomes

de armazenamento dos documentos onde a palavra surge. É a partir desta estrutura que são preenchidos o mapa de contadores de inserções do cliente e o índice de inserções do servidor. Esta classe possui ainda duas estruturas de dados que são necessárias para suportar as operações de pesquisa e de remoção: o mapa `originalFilenamesDecoder` que permite obter o nome original do ficheiro após a sua decifra na sequência de uma operação de pesquisa, e o mapa `index_filenames_decoder_` que permite mapear os nomes de todos os ficheiros introduzidos no sistema nos nomes de armazenamento que lhes foram atribuídos (necessário para dar suporte às operações de remoção). A seleção das palavras dos documentos é feita com recurso a um analisador disponível no repositório [44].

5.2.3 Classe Crypto

É a classe que implementa os métodos que cifram os ficheiros aquando da sua inserção no sistema e que fazem a sua decifra e verificação de integridade na sequência de uma pesquisa. Foi implementado um método que faz a decifra com a chave antiga e cifra com nova chave, mantendo o ficheiro sempre em memória, para dar suporte ao algoritmo de reindexação parcial e total(1).

5.2.4 Classe SophosClientRunner

Classe que foi adaptada para suportar as novas operações de inserção, pesquisa e remoção de ficheiros, bem como operações de reindexação parcial e total. É a classe que fornece à aplicação cliente os métodos que permitem comunicar com o servidor e solicitar dados ao mapa de contadores. É ainda responsável por instanciar as classes das mensagens gRPC e por solicitar as operações de cifra e decifra dos ficheiros à classe `Crypto`, que implementa as operações criptográficas. Para suportar a realização das operações de reindexação, esta classe passou a ter capacidade de gerir dois objetos da classe `SophosClient` em simultâneo.

5.2.5 Classe SophosClient

Esta classe faz a ligação entre o `SophosClientRunner` e os mapas de contadores do cliente. Quando o utilizador solicita uma operação ao sistema, esta classe é responsável por obter os dados necessários do mapa de contadores e realizar as operações criptográficas do esquema Sophos. Para suportar as operações de remoção, os objetos desta classe passaram a controlar dois mapas de contadores em simultâneo (duas instâncias da classe `RockDBCounter`).

5.2.6 Classe Sophos (*Stub*)

Classe que é gerada automaticamente a partir do ficheiro `sophos.proto`, utilizado para definir o serviço gRPC. Esta classe possui um *remote interface type* ou *stub* que permite ao cliente chamar os métodos definidos no serviço gRPC. Oferece também duas

interfaces abstratas para serem implementadas pelo servidor [39]. Esta classe foi estendida com novos métodos e mensagens para suportar as operações implementadas, que estão descritos em detalhe no ponto 5.2.10.

5.2.7 Servidor e Classe SophosServerRunner

A função `main()` no ficheiro `src/sysAppServer/server.cpp` permite iniciar o servidor e criar uma instância da classe `SophosServerRunner`.

A classe `SophosServerRunner` é responsável por iniciar o servidor gRPC, como descrito na documentação disponível em [39]. O contrutor desta classe recebe como argumentos os caminhos para as diretorias onde estarão os índices, o endereço IP e a porta onde o servidor estará em execução. Cria ainda uma instância de um objeto da classe `SophosImpl` através do método `RegisterService`.

5.2.8 Classe SophosImpl

Esta classe implementa os métodos do gRPC *Service*, definidos no ficheiro `sophos.proto`. Com a receção da mensagem de `setup` enviada pelo cliente, o servidor cria a diretoria onde será armazenado o índice de inserções, o índice de remoções e as chave públicas necessárias realizar as operações sobre os índices. Cria também uma instância do objeto `SophosServer`.

5.2.9 Classe SophosServer

Esta classe instancia os objetos que comunicam com o índice (que são instâncias da classe `RockDBWrapper`). Para suportar remoções, esta classe foi estendida por forma a controlar dois índices, o de inserções e o de remoções. Como referido no ponto 5.1.1, a implementação de base do Sophos efetua pesquisas em paralelo sobre o índice. A extensão feita para suportar remoções teve de integrar as pesquisas nos dois índices, permitindo continuar a tirar partido da paralelização já realizada. Assim, realizam-se primeiro as pesquisas em paralelo no índice de inserções e os resultados obtidos são colocados numa estrutura de dados em memória. Seguidamente, executam-se as pesquisas no índice de remoções também em paralelo, e cada nome cifrado encontrado é verificado na estrutura de dados permitindo determinar as remoções.

5.2.10 Comunicação entre Cliente e Servidor

No que respeita à comunicação entre o cliente e o servidor, optou-se por estender a implementação existente da *framework* gRPC para suportar as novas operações. Foram adicionados os respetivos métodos e tipos de mensagens ao ficheiro `sophos.proto`:

- upload: método que envia ficheiros cifrados para o servidor. Foi criada a mensagem `UploadRequestMessage` que contém o nome de armazenamento do ficheiro, a sua dimensão e o ficheiro cifrado;

- delete_entry: método que envia as entradas a remover de forma lógica do sistema, isto é, novas entradas a inserir no índice de remoções. Foi criada a mensagem DeleteEntryMessage que envia para o servidor o UT e o nome cifrado do ficheiro;
- bulk_delete_entry: método que permite enviar as entradas a eliminar do sistema de forma lógica utilizando uma sequência de mensagens. Utiliza as mensagens DeleteEntryMessage;
- reindex_setup: método que indica ao servidor que vai ser efetuada uma reindexação e envia a nova chave pública, que será utilizada para as entradas nos novos índices. A mensagem criada para este método é a `ReindexSetupMessage`;
- reindex_insert: método que permite enviar entradas para o novo índice de inserções, durante a realização de uma operação de reindexação. Utiliza a mensagem `ReindexUpdateRequestMessage`;
- bulk_reindex_insert: método que envia as entradas para o novo índice de inserções através de uma sequência de mensagens, durante a operação de reindexação. Utiliza também as mensagens `ReindexUpdateRequestMessage`;
- reindex_change_db: método que permite ao cliente informar o servidor que pode trocar os índices. No final de uma execução com sucesso, o espaço de armazenamento das remoções lógicas foi libertado. Utiliza as mensagens `ReindexChangeRequestMessage`;
- reindex_search: solicita operações de pesquisa sobre o índice durante operações de reindexação. Utiliza as mensagens `ReindexSearchRequestMessage`;
- download: método utilizado pelo cliente para solicitar um ficheiro específico do servidor. Utiliza as mensagens `DownloadRequestMessage`. Ficheiro envia a resposta utilizando mensagens `DownloadReply`;
- delete_file: mensagem enviada para o servidor a solicitar a remoção de um ficheiro do armazenamento. Utiliza a mensagem `DeleteFileRequestMessage`;
- download_all: mensagem que indica ao servidor que deve enviar todos os ficheiros que tem armazenados para o cliente. O servidor envia os ficheiros recorrendo a mensagens `DownloadReply`.

5.3 Implementação da Arquitetura "Apenas Armazenamento"

Nesta secção são descritas as duas implementações realizadas da arquitetura "apenas armazenamento". Primeiro é descrito o sistema que não utiliza nenhum tipo de componente auxiliar, alojando o índice e a base de dados no sistema de armazenamento S3, e depois o sistema que utiliza o SGBD RocksDB-Cloud [32] como *buffer* e *cache* para as

entradas dos índices. Ambos os sistemas realizam todas as computações na máquina cliente passando a instanciar localmente o cliente e o servidor, sendo este último aquele que efetua as comunicações com o serviço de alojamento S3. Todas as operações disponibilizadas pelo sistema que implementa a arquitetura "computação com armazenamento" são também disponibilizadas nestes sistemas.

5.3.1 Sistema sem *Buffer* e *Cache*

Neste sistema, que será referido como Sistema Apenas Armazenamento (SAA), o serviço de armazenamento S3 aloja os dois índices e a base de dados. A figura 5.3 apresenta um diagrama simplificado do sistema e, nos pontos seguintes, são detalhados os aspetos modificados em cada uma das classes, em relação ao SCA.

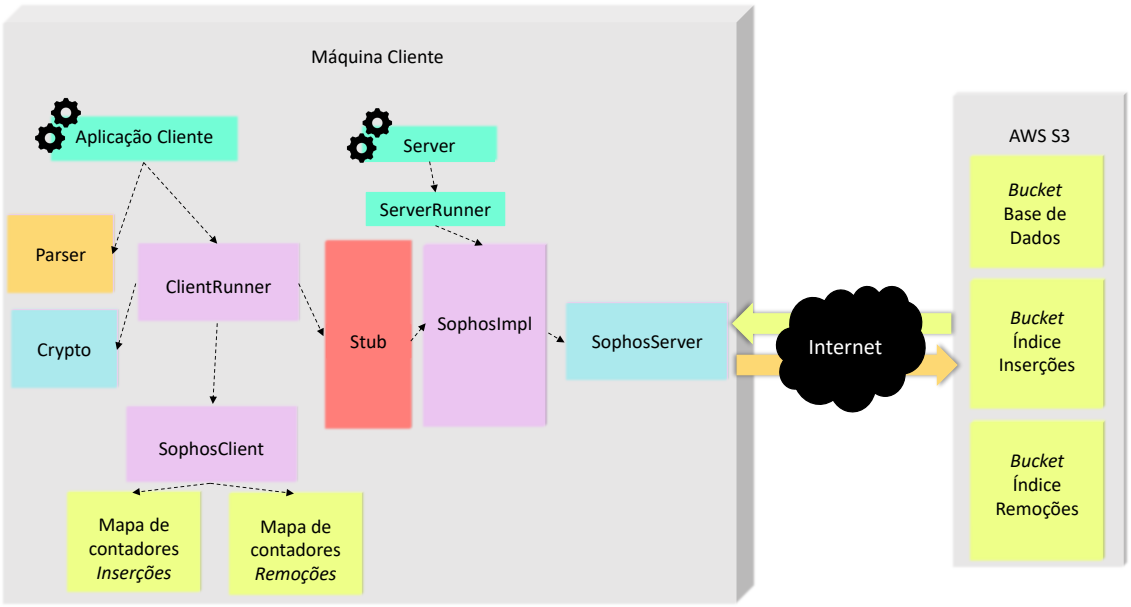


Figura 5.3: Diagrama simplificado do Sistema Apenas Armazenamento (SAA).

5.3.1.1 SophosClient

Esta classe realiza as operações sobre o mapa de contadores de inserções e de remoções mas sem utilizar o SGBD RocksDB, tendo sido feita uma nova implementação dos mapas e da forma como é assegurada a sua persistência entre sessões do cliente.

5.3.1.2 Servidor e Classe SophosServerRunner

O endereço onde o servidor está em execução é o *localhost*, opção definida no ficheiro de propriedades. Esta alteração faz com que todas as computações do sistema sejam realizadas na máquina cliente. É na função `main()` do Servidor que é inicializado o *Software*

Development Kit (SDK) do fornecedor de serviços AWS[45], que vai permitir realizar operações no serviço de armazenamento S3. É ainda criado o objeto `S3Client` e definida a região AWS que será utilizada.

O construtor da classe SophosServerRunner foi alterado para receber como argumentos a referência para o objeto `S3Client` (que permite realizar as operações no serviço S3) e o prefixo escolhido para os nomes dos *buckets* que serão criados no serviço de armazenamento (*bucket prefix*).

5.3.1.3 Classe SophosImpl

O construtor desta classe passou também a receber como argumentos a referência para o objeto `S3Client` e o *bucket prefix*. Continua a ser criada a diretoria `server.db` pois é necessário guardar em disco os ficheiros com as chave públicas do cliente (chaves do esquema de CP) mas deixam de ser utilizadas as instâncias da classe `RockDBWrapper`, uma vez que as entradas dos índices passarão a ser colocadas em *buckets* no serviço S3.

Os nomes dos *buckets* no serviço S3 têm de ser globalmente únicos e não é possível a sua alteração depois de criados. Por esta razão foi implementado o método `copyFilesOrder()` que permite copiar todos os elementos de um *bucket* para outro, para dar suporte à operação de reindexação parcial.

Foi alterado o método que permite finalizar a operação de reindexação, o método `reindex_change_db()`. Este método altera a diretoria onde é guardada a chave pública do esquema de CP e realiza as operações que apagam os *buckets* antigos, deixando o sistema pronto para futuras operações de reindexação.

Para que o sistema suporte multiplas sessões, foi adicionado um método que verifica os nomes dos *buckets* em utilização, criados em sessões anteriores. Esta informação permite que no reinício de uma sessão, o sistema reconheça que existem *buckets* válidos e sejam utilizados esses para lançar as instâncias da classe `SophosServer`.

Uma vez que neste sistema os ficheiros cifrados são guardados em *buckets* no serviço S3 e não no disco local da máquina, houve necessidade de criar um método que permita diferenciar as inserções de novos ficheiros das inserções realizadas nas operações de reindexação total. O método `reindex_upload()` recebe as mensagens com os novos ficheiros cifrados e insere-os no *bucket* dos ficheiros já processados na reindexação.

5.3.1.4 Classe SophosServer

Nesta classe foram criados dois construtores, um que será chamado na primeira utilização do sistema e que recebe um apontador para o objeto `S3Client`, as chaves públicas do esquema de CP e o prefixo a utilizar nos nomes dos *buckets* que serão criados, e um segundo construtor que recebe o apontador para o objeto `S3Client`, as chaves públicas do esquema de CP e os nomes dos *buckets* atuais a utilizar, já criados nas sessões anteriores. O primeiro construtor, através do prefixo que recebe, cria os *buckets* necessários à operação

do sistema: um *bucket* para as entradas do índice de inserções, um *bucket* para as entradas do índice de remoções e um *bucket* para o armazenamento dos ficheiros (ou base de dados). Para que os *buckets* criados possam ser utilizados em futuras sessões do sistema, este construtor chama o método `saveSessionData()` para guardar esta informação. Este método cria ficheiros no disco local da máquina e que serão lidos nos futuros inícios de sessão. O segundo construtor, uma vez que recebe os nomes dos *buckets* que precisa para operar, apenas inicializa as variáveis de instância da classe.

Todos os métodos de pesquisa desta classe foram adaptados para realizar as leituras e escritas nos respetivos *buckets* e foram adicionados métodos que permitem criar e apagar *buckets*, nomeadamente o método `createNewBucket()` e o método `deleteBuckets()`.

5.3.2 Sistema com *Buffer* e *Cache*

Ao utilizar-se um serviço de armazenamento externo como substituto de uma base de dados "chave-valor" serão feitos muitos pedidos nas operações que têm de verificar as entradas que pertencem ao índice. A utilização de uma *cache* no cliente permite reduzir o número desses pedidos.

O SGBD RocksDB-Cloud [32] foi desenvolvido a partir do RocksDB [31] e tem características que possibilitam a sua utilização como *buffer* e como *cache* dos índices de inserções e remoções nesta arquitetura. O SGBD RocksDB utiliza uma *Log Structure Merged Tree* para alojar os dados e a implementação RocksDB-Cloud permite que esta estrutura seja colocada no serviço de alojamento S3. O cliente armazena localmente alguma informação e utiliza uma *cache* de blocos, onde são armazenados os blocos dos ficheiros SST pesquisados recentemente. [31, 32, 46].

O código desta implementação partiu da implementação do SAA, uma vez que este sistema já utilizava a nova implementação do mapa de contadores sem o SGBD RocksDB. A figura 5.4 apresenta um diagrama simplificado deste sistema, designado por Sistema Apenas Armazenamento com *Buffer* e *Cache* (SAABC), e as alterações realizadas ao código encontram-se detalhadas no resto da secção.

5.3.2.1 Classe SophosServer

Esta classe instancia os objetos `RockDBWrapper` que materializam o índice de inserções e o índice de remoções e lida diretamente com o sistema de armazenamento S3 para armazenar os ficheiros. Os métodos que realizam operações sobre os índices mantém-se iguais aos utilizados no SCA. Os métodos que lidam com o armazenamento de ficheiros foram alterados para fazer escritas e leituras no *bucket* que contém a base de dados.

5.3.2.2 Classe RockDBWrapper

Esta classe foi adaptada para utilizar o SGBD RocksDB-Cloud e permitir que este coloque os seus ficheiros no sistema de armazenamento S3.

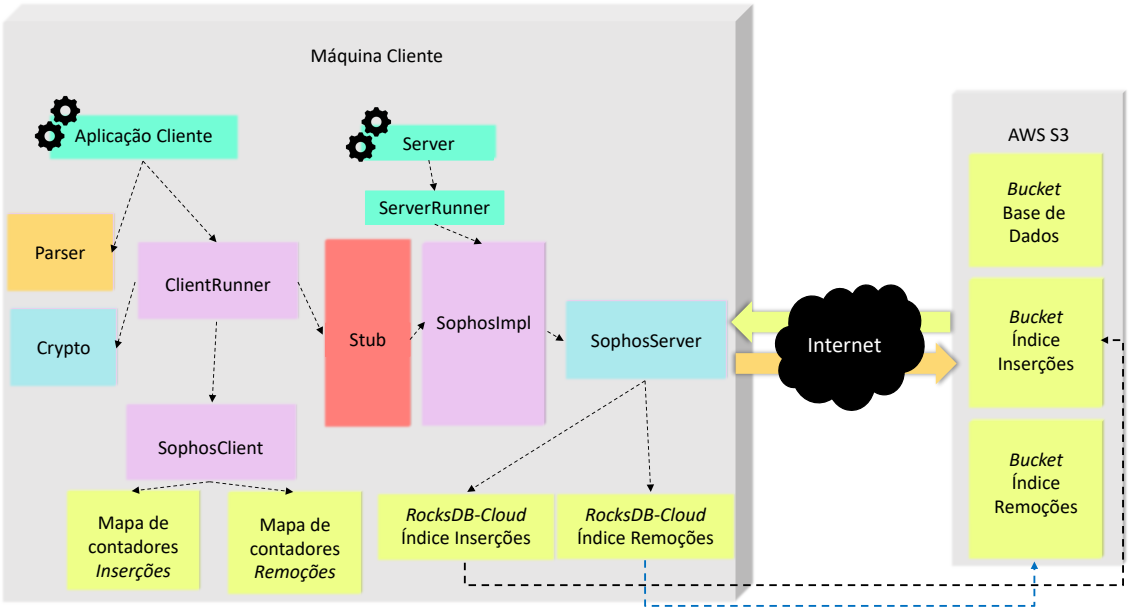


Figura 5.4: Diagrama simplificado do Sistema Apenas Armazenamento com *Buffer* e *Cache* (SAABC).

5.4 Armazenamento e Operações sobre Ficheiros (T1)

Nesta secção são apresentadas as decisões de implementação sobre o local de armazenamento da base de dados e das operações de tratamento e manipulação de ficheiros.

5.4.1 Onde são guardados os ficheiros?

Para fazer o alojamento da base de dados, o SCA utiliza a memória não volátil da VM e os sistemas que implementam a arquitetura "apenas armazenamento" utilizam o serviço AWS S3.

5.4.2 Cifra e Chaves Utilizadas

A cifra dos ficheiros é feita utilizando a construção AEAD XChaCha20-Poly1305 na versão *Internet Engineering Task Force* (IETF) [47]. Esta construção criptográfica cifra as mensagens com uma chave simétrica e um *nonce* e garante que, tanto a mensagem como os dados adicionais fornecidos (não cifrados) não foram alterados, calculando um *hash* ou *tag* [40]. Uma das principais características desta construção criptográfica é o tamanho do *nonce* utilizado, 192 *bits*. Pela sua dimensão, é possível e seguro gerar *nonces* de forma aleatória, o que permite libertar o sistema de ter de guardar esta informação no cliente. Cifra até (aproximadamente) 2^{64} mensagens com a mesma chave simétrica [40]. As restantes características desta construção criptográfica são apresentadas na tabela 5.1.

O cliente cria uma chave simétrica de 256 *bits* da primeira vez que é executado e essa chave é mantida pelo sistema em armazenamento não volátil até ser ordenada a realização

Tabela 5.1: Características da construção criptográfica utilizada para realizar a cifra dos ficheiros introduzidos no sistema [40].

Construção	Chave	Nonce	Bloco	MAC
XChaCha20-Poly1305-IETF	256 bits	192 bits	512 bits	128 bits

de uma operação de reindexação. Quando uma operação de reindexação total é realizada, é gerada uma nova chave simétrica que será utilizada na cifra dos ficheiros.

5.4.3 Inserção de Ficheiros

A realização da operação de inserção de ficheiros necessita que estes sejam colocados previamente na diretoria `/inputFolder`. O utilizador pode solicitar a inserção de um único ficheiro, indicando o seu nome, ou selecionar a opção que permite introduzir todos os documentos que se encontrem na diretoria. O objeto `FileParser` faz a extração das palavras-chave contidas no(s) ficheiro(s) e atribui o(s) nome(s) de armazenamento. Este "nome" é um número inteiro de 64 *bits* atribuído aleatoriamente a cada ficheiro novo introduzido no sistema. No final desta operação, o nome original do ficheiro e o seu nome de armazenamento estão guardados no mapa `originalFilenamesDecoder`, que mapeia o nome cifrado do ficheiro no seu nome original, e no mapa `index_filenames_decoder_`, que mapeia o nome original no seu nome cifrado.

Os sistemas não permitem a inserção de ficheiros duplicados com o mesmo nome original, mesmo quando inseridos em operações distintas.

Com o nome de armazenamento atribuído e extraídas as palavras-chave, é solicitado ao objeto `SophosClientRunner` a inserção do ficheiro no sistema. O objeto `Crypto` realiza as operações de cifra e o ficheiro está pronto para ser armazenado. No SCA, uma vez recebido o ficheiro no servidor, este é armazenado localmente. No caso dos sistemas que não utilizam computação na *cloud*, é a classe `SophosServer` a correr no dispositivo cliente que envia o ficheiro para o serviço de armazenamento S3.

Uma vez que todos os ficheiros estejam em armazenamento não volátil, o sistema vai iniciar o preenchimento do mapa de contadores de inserções e índice de inserções, concluindo a operação. O SCA faz o envio de todas as entradas do índice para o servidor na *cloud*, o SAA envia todas as entradas para o serviço S3 utilizando uma *pool* de *threads* e o SAABC insere as entradas na *memtable* da instância do RocksDB-Cloud que implementa o índice de inserções, funcionando como um *buffer*. Os diagramas de sequência da operação de inserção para o SCA estão ilustradas nas figuras 5.5 e 5.6.

5.4.4 Pesquisa de Ficheiros

Esta operação permite ao utilizador indicar uma ou várias palavras a pesquisar e receber os documentos decifrados onde as palavras se encontram presentes. A aplicação

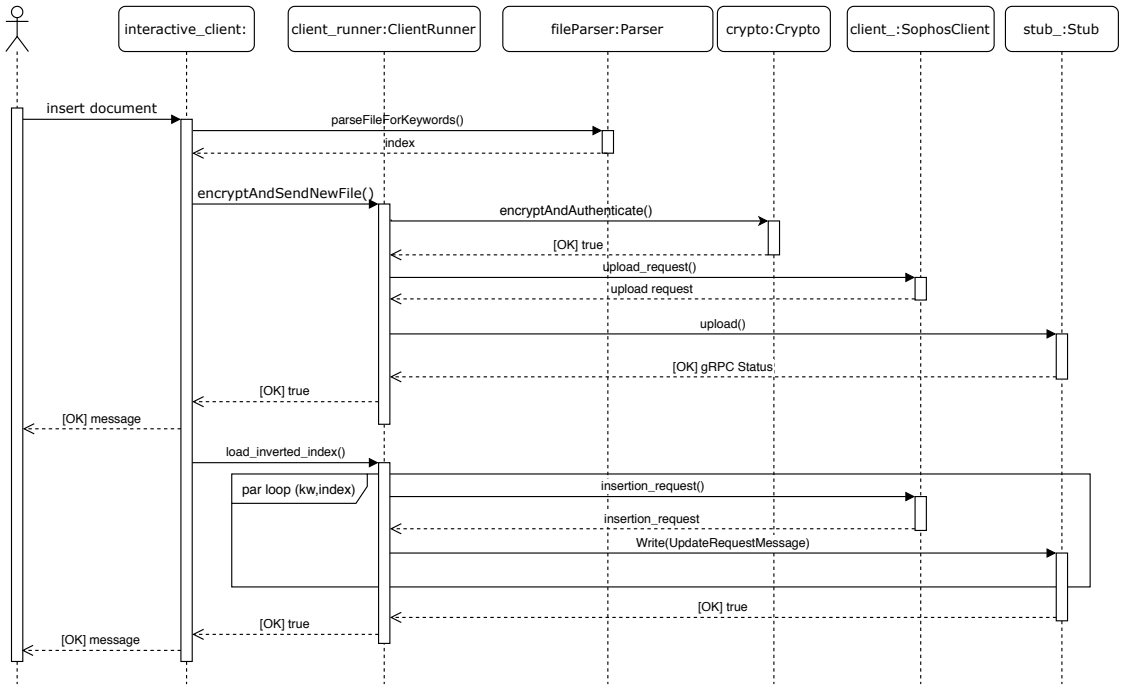


Figura 5.5: Diagrama de sequência simplificado da operação inserção de ficheiros no cliente, no Sistema Computação com Armazenamento (SCA).

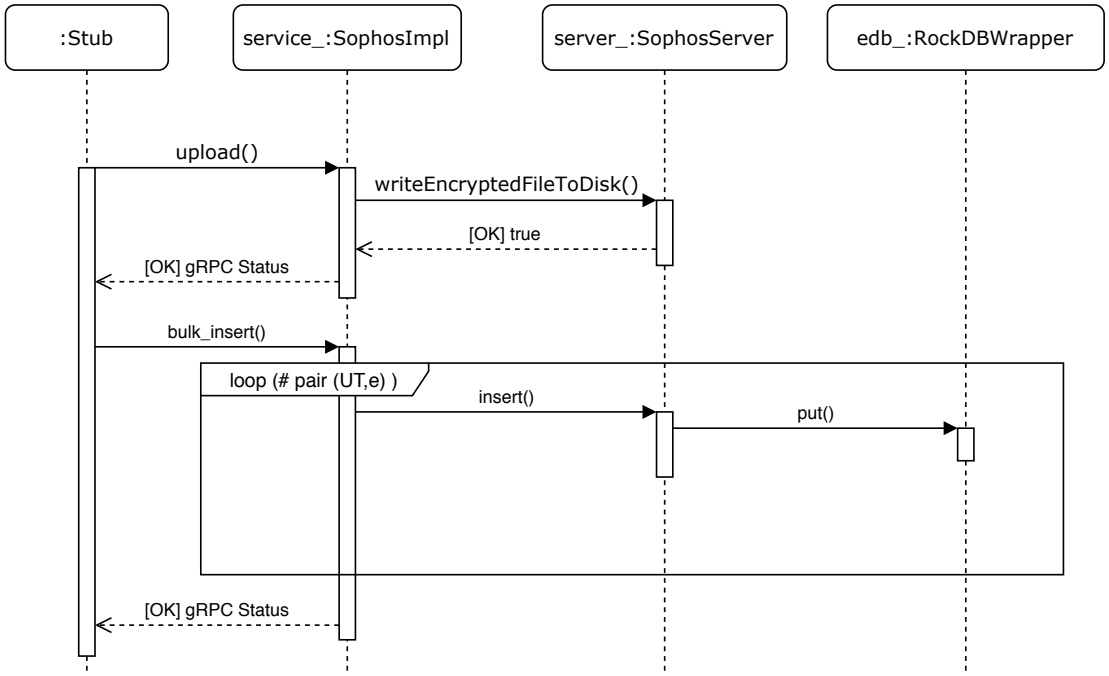


Figura 5.6: Diagrama de sequência simplificado da operação inserção de ficheiros no servidor, no Sistema Computação com Armazenamento (SCA).

cliente utiliza o método `search()` do `SophosClientRunner` para cada palavra-chave que o cliente indicou para pesquisa.

É gerada a chave a partir da palavra que se está a pesquisar e o *Search Token* (ST) correspondente ao valor do contador do mapa de contadores de inserções, como se pode verificar no pseudo-código da figura I.1. Como o sistema suporta remoções, é também necessário obter o valor do contador do mapa de remoções e o respetivo ST. O `SophosClientRunner` solicita esta informação ao objeto `SophosClient` e, uma vez na sua posse, utiliza o método `search()` do *Stub* do gRPC.

A partir deste ponto as computações a efetuar são as mesmas mas, dependendo do sistema, podem ser realizadas na VM a correr na *cloud* ou na máquina cliente. Uma vez recebido o pedido na classe `SophosImpl`, são calculados os *Update Tokens* (UTs) a partir dos STs e contadores recebidos. A partir dos UT são obtidos os nomes cifrados dos ficheiros e determinadas as remoções, permitindo devolver os ficheiros corretos ao utilizador. Nos casos em que o número do contador de inserções é superior a 2 (dois), a pesquisa sobre o índice de inserções é realizada em paralelo e os identificadores encontrados são colocados temporariamente numa estrutura de dados em memória. Posteriormente, é pesquisado o índice de remoções também em paralelo, e sempre que uma *thread* encontra um identificador de ficheiro, verifica se este já existe a estrutura de dados. Em caso afirmativo, a entrada é removida da estrutura. Esta implementação é uma extensão da implementação apresentada no ponto 5.1.1 e é possível pois o sistema não permite a inserção de dois ficheiros com o mesmo nome original nem com o mesmo nome de armazenamento. No final da pesquisa sobre o índice de remoções, a estrutura de dados só contém os nomes de armazenamento dos ficheiros que contêm a palavra chave pesquisada e que não foram removidos do sistema. No entanto, a forma de obter os identificadores dos ficheiros a partir dos UT e a forma de obter os ficheiros armazenados varia entre os sistemas:

- SCA: o servidor, para cada UT calculado, acede ao índice de inserções e remoções obtendo o nome cifrado do ficheiro a ele associado. Após filtrar os dados obtidos, identifica as operações de remoção já efetuadas e obtém os ficheiros que estão no armazenamento não volátil da VM;

- SAA: a máquina cliente, por cada UT de inserção e remoção calculado, faz um pedido ao *bucket* correspondente para obter o nome cifrado do ficheiro que lhe está associado. Depois de obtidos os nomes de armazenamento dos ficheiros e filtradas as remoções, solicita o ficheiro (ou ficheiros) ao *bucket* da base de dados;

- SAABC: a máquina cliente, por cada UT de inserção e remoção calculado, começa por verificar as entradas na *memtable* e posteriomente na *cache* de blocos. Caso o nome cifrado não seja encontrado nestas estruturas locais, é enviado um pedido ao serviço de armazenamento, que lê o ficheiro SST onde a entrada procurada se encontra. Uma vez filtrados os resultados, os ficheiros selecionados são solicitados ao serviço S3.

Cada documento recebido no `SophosClientRunner` vai ser decifrado e escrito para disco com o nome original. É utilizado o método `decryptAndAuthenticate()` do objeto `Crypto`, ficando o ficheiro disponível para o utilizador na diretoria `/searchedFiles`. Os nomes dos documentos são ainda apresentados no terminal.

As operações de pesquisa do SCA realizadas no cliente estão ilustradas na figura 5.7 e no servidor na figura 5.8.

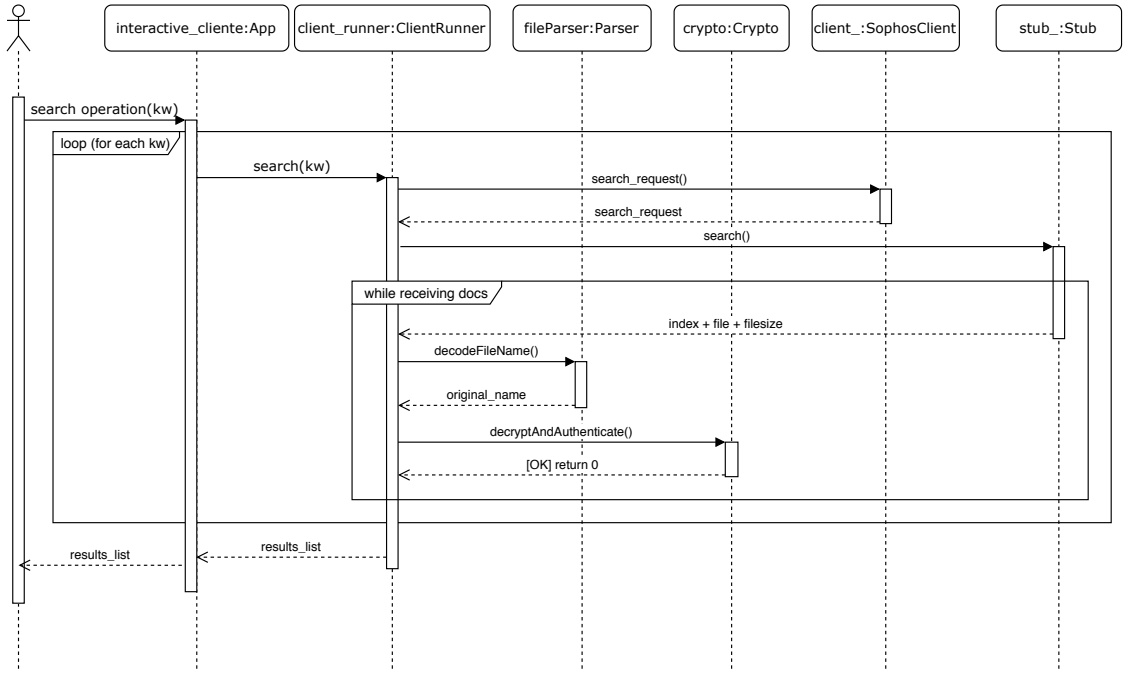


Figura 5.7: Diagrama de sequência simplificado da operação de pesquisa por palavra-chave no cliente, no Sistema Computação com Armazenamento (SCA).

Para evitar o envio de mensagens desnecessárias foi implementada uma verificação nos mapas de contadores, que permite identificar que uma palavra pesquisada não existe em nenhum documento no sistema. Esta verificação é possível uma vez que a diferença entre o contador de inserções e o contador de remoções, para uma determinada palavra, indica o número de documentos onde essa palavra existe. Se esse valor for igual a zero, significa que todas as inserções dessa palavra foram posteriormente removidas de forma lógica.

5.4.5 Remoção de Ficheiros

Para remover um ficheiro é necessário que o utilizador efetue antes uma operação de pesquisa. Os ficheiros retornados podem então ser removidos.

À semelhança da inserção, o utilizador pode indicar o nome em claro do ficheiro que será removido ou remover todos os objetos que se encontrem na diretoria `searchedFiles`. Para cada ficheiro indicado para remoção, o objeto `FileParser` encontra o nome cifrado

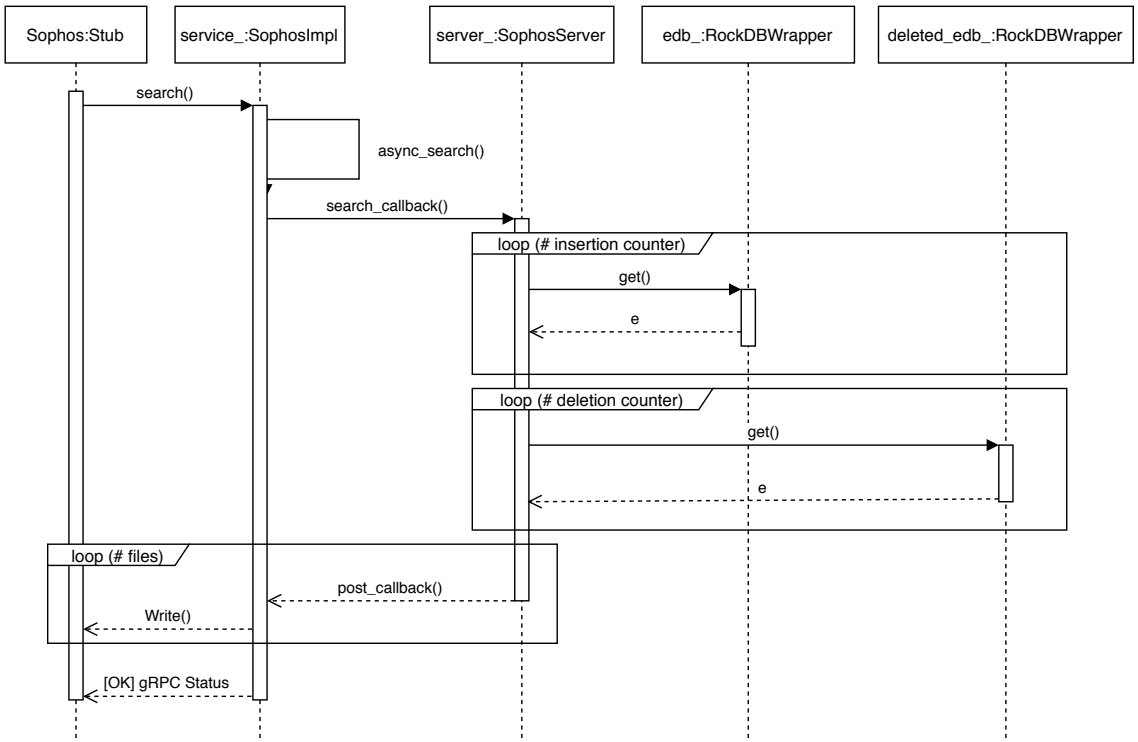


Figura 5.8: Diagrama de sequência simplificado da operação de pesquisa por palavra-chave no servidor, no Sistema Computação com Armazenamento (SCA).

que lhe corresponde e encontra todas as palavras-chave, para que possam ser introduzidas no índice de remoções. As palavras encontradas são mantidas temporariamente em memória, na estrutura de dados `invertedIndex`. Após esta sequência de ações pode solicitar-se a remoção física do ficheiro.

No SCA, o cliente começa por solicitar ao servidor a eliminação do ficheiro do local onde está armazenado e, após essa ação estar concluída, iniciam-se as operações de inserção dos pares (UT, nome cifrado) no índice de remoções. No caso dos sistemas "apenas armazenamento" é a máquina cliente que envia os pedidos de eliminação do ficheiro para o serviço S3. Posteriormente, o SAA envia as entradas do índice de remoções para o *bucket* respetivo utilizando uma *pool* de *threads* e o SAABC coloca as entradas na *memtable* da instância do RocksDB-Cloud que materializa o índice de remoções.

As operações de remoção de ficheiros no SCA estão ilustradas na figura 5.9 e na figura 5.10.

5.4.6 Necessidades de Persistência

O desenvolvimento das operações sobre ficheiros obrigou a que o cliente tivesse de guardar informações em armazenamento não volátil para suportar diferentes sessões. Os dados que têm de ser guardados nos três sistemas são aqueles que permitem converter os nomes dos ficheiros: os dados do mapa `originalFilenamesDecoder`

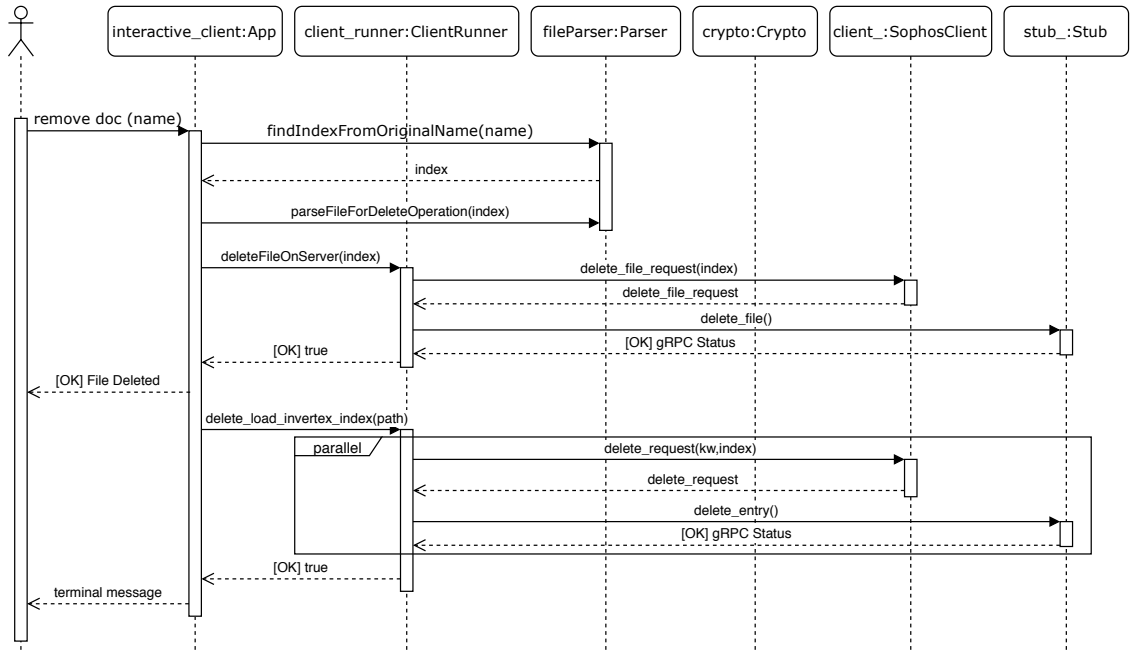


Figura 5.9: Diagrama de sequência simplificado da operação de remoção de ficheiros no cliente, no Sistema Computação com Armazenamento (SCA).

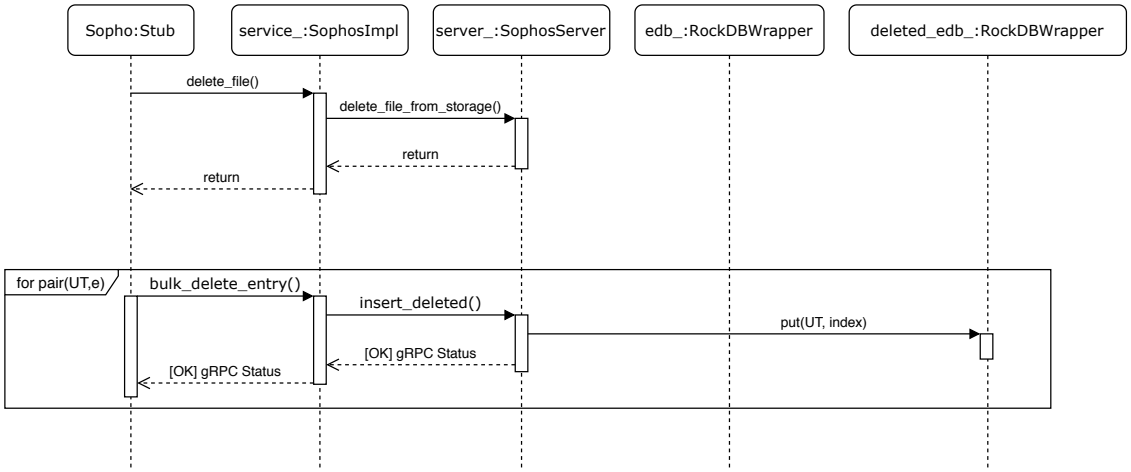


Figura 5.10: Diagrama de sequência simplificado da operação de remoção de ficheiros no servidor, no Sistema Computação com Armazenamento (SCA).

que mapeiam o nome de armazenamento no nome original, e os dados do mapa `index_filenames_decoder_` que mapeia os nomes originais nos nomes de armazenamento. A escrita destes dados para memória não volátil no cliente acontece no destrutor do objeto da classe `FileParser`. Quando se recupera o sistema, se os ficheiros que contêm estes dados existirem na pasta `sessionFiles`, é carregada a informação para as estruturas de dados em memória e o sistema pode ser utilizado, permitindo ao cliente recuperar a sua sessão.

Nos sistemas "apenas armazenamento", por não utilizarem o SGBD RocksDB para implementar o mapa de contadores, foi necessário desenvolver a funcionalidade que permite guardar a informação em disco entre as várias sessões. O carregamento dos mapas de contadores ao iniciar o sistema é feito no construtor do objeto `SophosServer`, através do método `loadCounterMapsFromDisk()`. Quando o utilizador termina a sessão, a informação é colocada em armazenamento não volátil utilizando o método `saveCounterMapsToDisk()`, chamado no destrutor da mesma classe. Como estes sistemas utilizam *buckets* no serviço S3, são ainda guardados em memória não volátil na máquina cliente os nomes dos *buckets* que estão em utilização. Esta informação é escrita para disco no construtor da classe `SophosServer` ao serem criados novos *buckets* e é recuperada no construtor da classe `SophosImpl`, quando o sistema reinicia uma nova sessão.

5.5 Reindexações e Libertação de Espaço no Índice (T3)

Os sistemas desenvolvidos suportam a operação de reindexação. Quando a operação apenas afeta os índices designa-se reindexação parcial, quando afeta os índices e a base de dados designa-se por reindexação total. Foram implementados dois algoritmos diferentes de reindexação total para testar diferentes abordagens, designados por reindexação total(1) e total(2). De salientar que as operações de reindexação implementadas apenas permitem reduzir espaço de armazenamento ocupado pelo índice e não do armazenamento de ficheiros. Os ficheiros apenas podem ser removidos do sistema através das operações de remoção de ficheiros.

No caso das duas primeiras reindexações referidas (parcial e total(1)), a sequência de operações é semelhante e a informação para preenchimento do novo índice de inserções é obtida através da pesquisa nos índices antigos, diferindo apenas no facto que a reindexação total cifra a base de dados com uma nova chave criptográfica simétrica. A reindexação total(2) implementa uma abordagem diferente, substituindo as pesquisas no índice por uma operação de *download* de todos os ficheiros que estão armazenados no sistema. Começaremos por descrever as operações de reindexação parcial e total(1) por partilharem a maior parte do código que as implementa e posteriomente será analisada a implementação da reindexação total(2).

Nas operações de reindexação parcial e total(1) são pesquisadas todas as palavras-chave que existem no sistema, para identificar os ficheiros que as contêm e voltar a inserir o par (*Update Token* (UT), nome cifrado do documento) no novo índice. A aplicação cliente,

através do método `getAllKeywordsToSearch()` do objeto `SophosClientRunner`, acede ao mapa de contadores para obter a lista de palavras-chave que têm de ser pesquisadas no índice. O objeto `SophosClient` preenche uma lista de palavras-chave acedendo ao mapa de contadores do cliente e coloca também todas as entradas do mapa de contadores de remoções numa *hash table*, podendo calcular as palavras que já não existem em nenhum documento da base de dados. Esta operação permite reduzir o número de mensagens de pesquisa trocadas entre o cliente e o servidor no SCA ou reduzir o número de pedidos ao serviço de armazenamento S3 no caso do SAA e do SAABC.

Uma vez na posse de todas as palavras-chave a pesquisar, é necessário criar as novas estruturas para a realização da operação de reindexação. As estruturas criadas dependem do sistema considerado e são:

- SCA: criação dos novos mapas de contadores no cliente e novos índices cifrados no servidor a executar na VM, ambos recorrendo ao SGBD RocksDB, e criação de novas chaves criptográficas para o esquema de CP;
- SAA: criação dos novos mapas de contadores na máquina cliente, criação dos novos *buckets* para os índices e base de dados no serviço S3 e, por fim, as chaves criptográficas novas do esquema de CP;
- SAABC: criação dos novos mapas de contadores, novos índices utilizando o SGBD RocksDB-Cloud na máquina cliente, novo *bucket* no serviço S3 para a base de dados e as novas chaves criptográficas do esquema de CP.

A aplicação cliente solicita a realização destas operações através da chamada do método `createNewKeysAndClientMaps()` do objeto `SophosClientRunner`. Este método faz uma chamada do método estático `init_reindex_client_in_directory()` que vai criar todas as estruturas de dados e chaves necessárias ao lançamento de um novo objeto da classe `SophosClient`, que fará as operações sobre os mapas de contadores agora criados. O objeto `SophosClientRunner` envia uma `ReindexSetupMessage` que, ao ser recebida pelo objeto `SophosImpl`, dá início às operações de criação de um novo objeto `SophosServer` e dos novos índices. As operações iniciais da reindexação no cliente do SCA encontram-se ilustradas na figura 5.11.

Neste momento da execução existem no sistema dois objetos `SophosClient` e dois objetos `SophosServer`. O cliente inicia as pesquisas por todas as palavras-chave que filtrou, recebendo os nomes de armazenamento dos documentos que ainda existem no sistema e que contêm essas palavras-chave. Terminadas todas as pesquisas, iniciam as inserções das entradas no novo índice de inserções. Caso se trate de uma reindexação total, é nesta altura que será realizado o pedido dos ficheiros ao servidor e a cifra com a nova chave.

Para terminar têm de ser realizados os dois últimos passos: é necessário efetuar a destruição das estruturas antigas (quer dos mapas de contadores do cliente, quer dos índices)

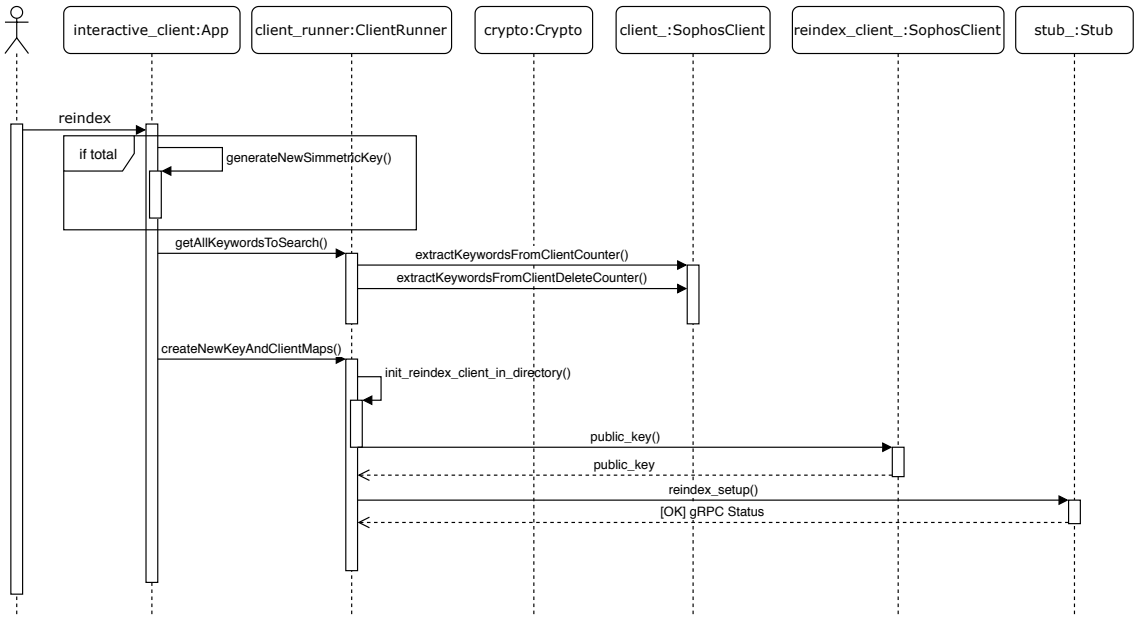


Figura 5.11: Diagrama de sequência simplificado do início da operação de reindexação no cliente no Sistema Computação com Armazenamento (SCA)

e as operações de gestão das chaves criptográficas simétricas. Para realizar as operações sobre as estruturas de dados, a aplicação cliente chama o método `changeDefaultDB()` para iniciar a operação. Este método cria a mensagem `ReindexChangeRequestMessage` que, ao ser recebida pelo objeto `SophosImpl`, permite trocar a instância `SophosServer` que vai passar a ser utilizada no sistema. Uma vez esta troca efetuada, as chaves criptográficas do Sophos, o índice de inserções e o índice de remoções mais antigos são eliminados. O mesmo procedimento é feito para os objetos `SophosClient`, sendo eliminados os mapas de contadores mais antigos. No que respeita às operações sobre as chaves criptográficas utilizadas para a base de dados, o cliente altera o nome da chave simétrica antiga para `old_sym_key_256.key` e da chave mais recente para `sym_key_256.key`, deixando o sistema preparado para que, numa futura reindexação, possa ser criada uma nova chave criptográfica simétrica, que será criada com o nome `new_sym_key_256.key`. A sequência de operações descritas, para o SCA, estão ilustradas no diagrama de sequência da figura 5.12.

A reindexação total(2) executa também as operações iniciais de criação da nova chave criptográfica simétrica e das novas estruturas de dados bem como as mesmas operações finais sobre as estruturas de dados criadas e sobre as chaves. As diferenças na implementação estão na forma como o cliente consegue aceder à informação que efetivamente ainda existe no sistema, ou seja, às entradas do índice que não foram apagadas na sequência da remoção de ficheiros. Ao contrário das operações de reindexação anteriormente descritas, nesta implementação essa informação é obtida a partir dos ficheiros que ainda existem no sistema, uma vez que o cliente solicita o envio de todos os ficheiros da base de dados. No caso do SCA esse pedido é feito ao servidor que está a executar a VM. No caso dos

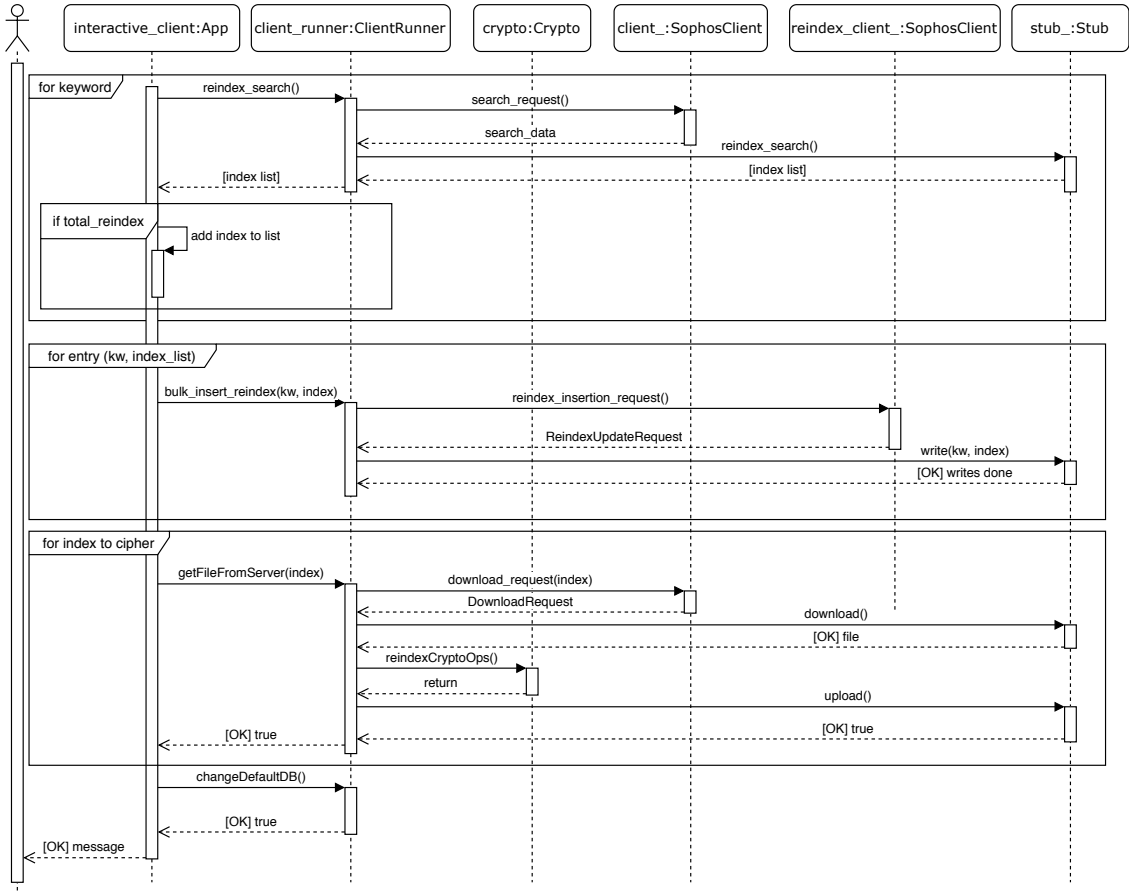


Figura 5.12: Diagrama de sequência simplificado das operações de reindexação parcial e total (1) no cliente, no Sistema Computação com Armazenamento (SCA).

sistemas "apenas armazenamento", a máquina cliente obtém todos os ficheiros que estão armazenados no *bucket* da base de dados. O pedido de obtenção dos ficheiros é feito pelo método `startExperimentalSystemReindex()` do `SophosClientRunner` que, por sua vez, executa o método `download_all()` do *Stub*. Cada ficheiro recebido na máquina cliente é decifrado com a chave antiga e é realizada uma nova extração das palavras-chave. Volta a ser cifrado com a nova chave criptográfica simétrica e colocado de novo na solução de armazenamento. Terminadas as operações sobre os ficheiros, o cliente inicia as operações de colocação das entradas (palavra-chave, nome cifrado) no novo mapa de contadores de inserções e índice de inserções.

5.6 Tratamento dos Nomes Originais dos Ficheiros (T4)

A transformação dos nomes originais dos ficheiros implica considerar dois aspetos. O primeiro é o sistema operativo que será utilizado nos componentes que efetuam computações, pois existem caracteres que não são permitidos e não são transversais a todos os sistemas. O segundo aspeto são as operações criptográficas que serão realizadas sobre os nomes depois de alterados e podem condicionar o número de *bits* que poderão ter. Foram

consideradas várias abordagens possíveis para esta questão. No entanto, uma vez que a implementação do Sophos utiliza uma função HMAC que espera nomes de ficheiros de 64 *bits*, trabalhou-se no sentido de encontrar uma solução para nomes desta dimensão.

A solução inicialmente considerada foi a cifra dos nomes dos ficheiros. No entanto, os caracteres que resultam desta cifra podem pertencer ao conjunto de caracteres proibidos ou reservados pelo sistema operativo, o que não permite a utilização desta solução.

Foi também considerada a utilização de uma função de síntese, que geraria o nome de armazenamento a partir do conteúdo do ficheiro ou do seu nome original. Esta opção tinha o problema de que, para evitar colisões, o número de *bits* gerado pela função de síntese teria ser elevado, idealmente igual ou superior a 128 *bits* [48]. Este facto levaria a que fosse utilizado mais espaço de armazenamento no índice, teria de ser suportado pelas operações criptográficas que trabalham com os nomes dos ficheiros e teria de se garantir que o resultado da função de síntese não seria superior à máxima dimensão suportada pelos sistemas operativos, para os nomes dos ficheiros. Para o caso específico dos sistemas desenvolvidos, como o objetivo era encontrar uma solução que gerasse nomes de armazenamento de 64 *bits*, esta opção também não foi utilizada.

Outra opção seria substituir o nome original do ficheiro por um número inteiro e incrementar esse valor a cada nova inserção. Esta solução evitaria a possibilidade de relacionar o nome de armazenamento do ficheiro com o seu nome original e permitiria utilizar nomes de 64 *bits*. A desvantagem encontrada seria o facto de se estar a revelar ao servidor a ordem temporal pela qual os documentos foram cifrados no cliente. No entanto, durante o desenvolvimento do sistema, como se tornou necessário guardar a informação dos nomes originais dos ficheiros e o nome de armazenamento que lhe foi atribuído, foi possível dispensar a atribuição dos nomes de armazenamento a partir de um contador e passou-se a utilizar números aleatórios, pois a cada inserção de um novo ficheiro, consegue-se facilmente verificar se o número selecionado já foi atribuído anteriormente. Assim, escolheu-se para método de atribuição dos nomes cifrados a geração aleatória de números inteiros de 64 *bits*, do tipo `unsigned long long`, que permitem valores entre 0 e 18.446.744.073.709.551.615, garantindo números suficientes para lidar com inserção de vários ficheiros. O elemento do sistema responsável pela atribuição dos novos nomes aos ficheiros cifrados garante o tratamento em caso de seleção de dois números iguais.

5.7 Notas Finais de Capítulo

Neste capítulo foi feita uma apresentação do código que serviu de base para a construção dos três sistemas e detalhadas as alterações realizadas em cada um deles. Foram descritas as implementações das operações suportadas, que permitiram aprofundar o estudo dos tópicos 1 e 3, e da transformação dos nomes dos ficheiros inseridos, que responde ao problema identificado no tópico 4.

No próximo capítulo serão apresentados os resultados dos testes de performance, os modelos de custos e a análise de segurança de cada um dos sistemas desenvolvidos.

Capítulo 6

Avaliação e Análise das Soluções, Modelo de Custos e Discussão de resultados

A implementação das arquiteturas descritas no capítulo 4 permitiu aprofundar o estudo de alguns dos pontos analisados no capítulo 3, como o tratamento dos ficheiros nas operações e a forma de transformar os seus nomes. No entanto, é necessário testar as soluções encontradas para confirmar a viabilidade dos sistemas. Neste capítulo serão apresentados os resultados da avaliação de performance, os modelos de custos e análise de segurança dos sistemas. Uma vez que os créditos disponíveis para testes estavam limitados, as medições de performance para o Sistema Apenas Armazenamento (SAA) não cobriram todas as operações implementadas nem todos os conjuntos de teste.

Para a realização dos testes foram obtidos sub-conjuntos de ficheiros `.txt` da *Wikipedia* na lingua inglesa[1], utilizando o projeto WikiExtractor[2]. As características dos conjuntos estão listadas na tabela 6.1, onde se indica o nome atribuído ao conjunto, os pares (palavra-chave, nome do documento) extraídos dos ficheiros, o tamanho total do conjunto de ficheiros (em claro), o número total de ficheiros que compõem o conjunto e o tamanho médio dos ficheiros.

Os testes foram realizados utilizando como cliente uma máquina com o Sistema Operativo (SO) MAC OS Catalina, com um processador Dual-Core Intel i7 2,8 GHz e com 8GB de memória RAM. O sistema que implementa a arquitetura "computação com armazenamento" foi testado utilizando uma instância *t2.medium* no serviço AWS S3 com 2 *Virtual Central Processing Unit* (vCPU), 4 GB de memória e 40 GB de armazenamento não volátil [49], com o SO Ubuntu Server 16.04 LTS e localizada na região *US East (Northern Virginia)*. Os sistemas que implementam a arquitetura "apenas armazenamento" utilizaram o serviço de armazenamento *Simple Storage Service* (S3) do fornecedor de serviços

[1] https://en.wikipedia.org/wiki/Wikipedia:Database_download
[2] https://github.com/attardi/wikiextractor

Tabela 6.1: Conjuntos de dados utilizados na realização dos testes.

Conjunto	# Pares (k,i)	Tamanho(MB)	# Ficheiros	Tamanho (≈ p/ ficheiro)
A	995 171	63.72	62	1 MB
B	1 996 118	128.37	125	1 MB
C	3 965 288	256.05	249	1 MB
D	7 979 919	512.00	497	1 MB

AWS, também na região *US East (Northern Virginia)*. A localização dos recursos e o tipo de instância utilizada foram determinados pelas condições impostas pelo fornecedor de serviços, em virtude do tipo de conta disponível para realização dos testes.

6.1 Armazenamento e Operações sobre Ficheiros (T1)

A solução escolhida para fazer o alojamento de ficheiros condiciona a performance do sistema pois todas as operações implementadas acedem aos ficheiros cifrados que se encontram em memória não volátil. No caso do SCA, a avaliação realizada incidiu sobre o armazenamento de ficheiros na memória não volátil da VM onde o servidor está em execução. No SAA e no SAABC os ficheiros foram armazenados em *buckets*, no serviço S3.

Estudaram-se as necessidades temporais e espaciais das operações que manipulam ficheiros, nomeadamente a inserção, pesquisa e remoção e foi medido o volume e dimensão das mensagens trocadas entre o cliente e o servidor no SCA e entre o cliente e o serviço S3 no SAA e no SAABC. Os resultados do SAA foram obtidos apenas para o conjunto "A" devido à limitação dos créditos disponíveis.

6.1.1 Inserção de Ficheiros

A inserção de ficheiros é composta por dois conjuntos de sub-operações distintas: o primeiro sub-conjunto é responsável pela extração das palavras-chave, cifra dos ficheiros e o seu envio para a solução de armazenamento. O segundo sub-conjunto de operações coloca os pares (palavra-chave, documento) no mapa de contadores e no índice.

O tempo de execução das sub-operações foi medido em separado e os resultados estão na tabela 6.2. É apresentado o tempo de inserção de ficheiros, o tempo de inserção dos pares (palavra-chave, documento) no mapa de contadores e índice e o tempo total, todos eles medidos na máquina cliente. O tempo de inserção dos pares (UT, nome cifrado) é medido no elemento que opera sobre o índice cifrado, ou seja, no servidor para o SCA e na máquina cliente no SAA e no SAABC. O tempo de inserção do SAABC engloba o tempo de criação e envio do ficheiro SST para o *bucket* do índice de inserções.

O sistema que apresenta tempos de inserção menores é o SCA, uma vez que todos os dados se consideram no servidor assim que chegam à VM e são colocados no SGBD

Tabela 6.2: Tempos de execução da operação de inserção.

Conjunto	Sub-Op	SCA [min]	SAA [min]	SAABC [min]
A	Inserção Ficheiros	2.47	4.22	2.46
	Inserção (palavra,doc)	23.47	1108.91	26.41
	Inserção (UT, nome cifrado)	23.46	1108.85	26.39
	Total (Cliente)	25.95	1113.13	29.94
B	Inserção Ficheiros	4.42	-	5.03
	Inserção (palavra,doc)	50.35	-	60.73
	Inserção (UT, nome cifrado)	50.33	-	60.70
	Total (Cliente)	54.77	-	65.75
C	Inserção Ficheiros	8.47	-	9.89
	Inserção (palavra,doc)	100.94	-	117.88
	Inserção (UT, nome cifrado)	100.90	-	117.84
	Total (Cliente)	109.41	-	127.77
D	Inserção Ficheiros	17.83	-	19.40
	Inserção (palavra,doc)	212.68	-	270.18
	Inserção (UT, nome cifrado)	212.59	-	269.14
	Total (Cliente)	230.51	-	289.58

RocksDB. No SAABC os dados do índice apenas são colocados no serviço de armazenamento após a criação do ficheiro SST e envio para o *bucket* no serviço S3, pelo que este tempo foi contabilizado nas medições deste sistema. O SAA é o sistema que apresenta tempos de inserção mais elevados, sobretudo na inserção das entradas do índice. Esta duração de execução deve-se ao facto de cada entrada no índice de inserções ser um pedido de armazenamento ao serviço S3. Isto revela que, apesar de ser possível utilizar este sistema, os tempos de execução não são adequados para um sistema real.

No que respeita ao volume de dados enviados do cliente para fornecedor de serviços, todos os sistemas enviam o mesmo volume de ficheiros cifrados mas valores diferentes das entradas a colocar no índice de inserções. A tabela 6.3 apresenta o número total de palavras-chave retiradas dos documentos do conjunto (que vão dar origem aos pares (UT, nome cifrado)), o volume ocupado pelo envio desses pares para alojamento no índice e o volume do conjunto de ficheiros enviado, após a operação de cifra. No SCA os volumes foram obtidos através da medição das mensagens gRPC enviadas do cliente para o servidor onde se encontra a VM. No SAA o volume das entradas a colocar no índice é menor pois já não são utilizadas as mensagens gRPC [39] para as colocar no *bucket* do serviço S3. Por último, no SAABC, o volume de entradas a colocar no índice é o volume do ficheiro SST enviado para armazenamento no serviço S3.

6.1.2 Pesquisa de ficheiros

O objetivo principal de um sistema que utilize um esquema de CP é permitir ao utilizador a pesquisa dos ficheiros cifrados. Importa assim perceber a performance dos

Tabela 6.3: Volume de dados das mensagens enviadas durante a operação de inserção dos ficheiros.

Conjunto		SCA	SAA	SAABC
A	# Pares (UT, index)	995 171		
	Volume Pares (MB)	28.36	23.88	33.7
	Conjunto Cifrado (MB)	63.7		
B	# Pares (UT, index)	1 996 118		
	Volume Pares (MB)	56.88	-	67.01
	Conjunto Cifrado (MB)	128.4		
C	# Pares (UT, index)	3 965 288		
	Volume Pares (MB)	112.99	-	132.86
	Conjunto Cifrado (MB)	256.1		
D	# Pares (UT, index)	7 979 919		
	Volume Pares (MB)	227.40	-	266.61
	Conjunto Cifrado (MB)	512.0		

sistemas na realização das pesquisas e qual a influência que o suporte para remoções tem sobre essas operações.

Para cada conjunto pesquisaram-se as cinco palavras mais comuns da língua inglesa[3] que cumpriam os requisitos para serem palavras-chave e que estavam presentes em todos os documentos. As palavras selecionadas foram: *word*, *time*, *long*, *make* e *number*. A tabela 6.4 apresenta a média dos tempos obtidos nas pesquisas das cinco palavras, divididas por sistema e por localização de entradas do índice (coluna "Zona"). Nesta tabela *II* representa índice de inserções e *IR* representa índice de remoções. Estão presentes os valores dos tempos de pesquisa sobre as entradas do índice quando estas estão na *active memtable*, estrutura que guarda as entradas logo após a inserção e que funciona como *buffer*, e posteriormente os tempos de pesquisa quando as entradas do índice já estão colocadas em ficheiros SST, após a operação de *flush*, quer para o caso em que nenhuma remoção foi realizada no sistema, quer para o caso em que apenas um documento ficou armazenado. Apresenta-se ainda o tempo de obtenção dos ficheiros da solução de armazenamento, que no caso do SCA é o tempo que o servidor leva a obter o ficheiro do serviço *Elastic Block Store* (EBS) e a enviar para o cliente e no SAA e no SAABC é o tempo que o componente do servidor (a correr na máquina cliente) demora a obter o ficheiro do serviço S3. O SAA, por não utilizar o SGBD RocksDB, tem as entradas dos índices sempre colocadas no serviço S3 e por isso apresenta nomes diferentes na coluna "Zona".

Comparando os resultados obtidos, os tempos de pesquisa sobre o índice de inserções são semelhantes no SCA e SAABC quando as leituras são feitas a partir da *active memtable*.

[3] https://world-english.org/english500.htm

No entanto, quando as leituras são feitas a partir dos ficheiros SST, o SAABC apresenta tempos de pesquisa superiores pois, para cada chave procurada, o cliente tem de solicitar o bloco do ficheiro SST ao serviço de armazenamento S3 enquanto o SCA acede aos blocos no armazenamento da VM. Comparando o SAABC com o SAA, quando estes acedem às entradas do índice colocadas nos *buckets*, verifica-se que o segundo apresenta tempos de pesquisa menores pois, neste caso, é mais rápido pesquisar entradas de índice do que blocos de ficheiros SST, quando nenhum existe na *cache*.

No que respeita ao tempo total da operação de pesquisa, o SCA é mais rápido que o SAA e que o SAABC a executar as operações testadas.

As remoções vão obrigar a realizar mais operações pois as entradas do índice de remoções têm de ser verificadas. Uma vez que apenas um ficheiro foi mantido no sistema, os índices de inserções e remoções têm aproximadamento mesmo número de entradas, o que explica os tempos de pesquisa semelhantes sobre estas estruturas. No entanto, o aumento de entradas nos índices não afeta negativamente o tempo total das operações, uma vez que diminui o volume de ficheiros que têm de ser devolvidos ao cliente e as pesquisas são mais rápidas a devolver resultados. O SCA continua a apresentar tempos totais de execução das operações de pesquisa mais rápidos após as remoções.

6.1.3 Remoção de Ficheiros

No que respeita às operações de remoção, os tempos de colocação das remoções lógicas no índice de remoções são semelhantes aos tempos da operação de inserção pois é a mesma operação mas realizada sobre um índice diferente. O tempo total da operação é mais baixo uma vez que não existe o envio do ficheiro, apenas é enviada uma mensagem do cliente para o servidor a solicitar a sua remoção. A operação de remoção testada consistiu na remoção de todos os ficheiros armazenados com exceção de 1 (um). O ficheiro que permaneceu armazenado continha todas as palavras pesquisadas.

Analisando a tabela 6.5 confirma-se que a operação de remoção física dos ficheiros demora menos tempo que a operação que faz a sua inserção (ver tabela 6.2), sendo a maioria do tempo da operação consumida pela inserção das entradas no índice de remoções.

6.1.4 Requisitos de Armazenamento

Importa avaliar as necessidades de armazenamento dos vários elementos que compõem os sistemas. Nas estruturas que utilizam o SGBD RocksDB e RocksDB-Cloud, as maiores necessidades de armazenamento são atingidas quando as entradas dos índices estão ainda nas *memtables*. Após a operação de *flush*, o espaço ocupado pelos ficheiros SST é menor e, no caso do SAABC, esses ficheiros são armazenados no serviço S3 não ocupando espaço na máquina cliente.

Para o SCA, após a inserção dos conjuntos utilizados para realizar os testes e sem ter sido realizada qualquer remoção de documentos, os valores de armazenamento não volátil ocupado são os que se apresentam na tabela 6.6. Os valores apresentados indicam

Tabela 6.4: Tempos da operação de pesquisa (em segundos).

Sistema	Conjunto	Zona	Pesquisa II	Pesquisa IR	Obt Arm	Total
SCA	A	Memtable (srv)	0.010	0.0	10.44	10.94
		SST (srv)	0.012	0.0	9.50	10.00
		SST (del) (srv)	0.011	0.011	1.06	2.35
	B	Memtable (srv)	0.019	0.0	24.17	24.76
		SST (srv)	0.023	0.0	20.90	21.58
		SST (del) (srv)	0.022	0.022	0.22	0.98
	C	Memtable (srv)	0.036	0.0	38.896	39.427
		SST (srv)	0.042	0.0	41.243	42.714
		SST (del) (srv)	0.043	0.0418	0.1006	1.938
	D	Memtable (srv)	0.068	0.0	50.874	51.331
		SST (srv)	0.084	0.0	55.305	55.733
		SST (del) (srv)	0.085	0.085	0.21	1.081
SAA	A	*Bucket* S3	2.84	0.0	27.53	30.50
		Bucket S3 (del)	2.99	2.49	0.89	6.40
SAABC	A	Memtable (cli)	0.016	0.0	27.92	27.95
		SST (S3)	3.96	0.0	29.29	33.27
		SST (del) (S3)	3.47	2.90	1.36	7.77
	B	Memtable (cli)	0.021	0.0	57.85	57.90
		SST (S3)	5.15	0.0	57.865	63.04
		SST (del) (S3)	5.41	4.96	1.32	11.71
	C	Memtable (cli)	0.054	0.0	114.85	114.92
		SST (S3)	9.968	0.0	116.52	126.52
		SST (del) (S3)	10.56	9.51	1.318	21.41
	D	Memtable (cli)	0.095	0.0	241.94	242.05
		SST (S3)	18.432	0.0	228.30	246.76
		SST (del) (S3)	18.32	18.27	1.32	37.94

Tabela 6.5: Tempos de execução da operação de remoção.

Conjunto	Sub-Op	SCA [min]	SAA [min]	SAABC [min]
A	Remoção de Ficheiros	0.88	1.17	1.31
	Inserção (palavra,doc)	24.02	1106.08	26.81
	Inserção (UT,index)	24.02	1106.08	26.81
	Total (Cliente)	24.90	1107.27	28.12
B	Remoção de Ficheiros	1.81	-	1.96
	Inserção (palavra,doc)	51.86	-	57.10
	Inserção (UT,index)	51.85	-	57.10
	Total (Cliente)	53.67	-	59.06
C	Remoção de Ficheiros	3.77	-	3.72
	Inserção (palavra,doc)	103.82	-	117.42
	Inserção (UT,index)	103.80	-	116.35
	Total (Cliente)	107.60	-	121.14
D	Remoção de Ficheiros	7.00	-	7.89
	Inserção (palavra,doc)	216.13	-	260.9
	Inserção (UT,index)	216.10	-	260.35
	Total (Cliente)	223.12	-	268.79

Tabela 6.6: Volume ocupado pelas estruturas do Sistema Computação com Armazenamento (SCA) para cada conjunto testado, antes da operação de *flush* e sem remoções (Valores em MB).

Conjunto	Mapa de Contadores	Índice	Base de Dados	Sessão
A	33.6	45.8	63.7	0.0047
B	67.3	91.9	128.4	0.0094
C	133.5	141.1	256.1	0.0190
D	268.2	283.8	512.0	0.0381

as necessidades de armazenamento do cliente para os mapas de contadores e os ficheiros de sessão, e do servidor para o índice e base de dados. O cliente aloja ainda os ficheiros que contêm as chaves criptográficas necessárias ao funcionamento do esquema de CP, que totalizam 3478 *bytes* e as chaves criptográficas simétricas utilizadas na cifra de ficheiros (antiga e atual caso tenha já sido realizada uma reindexação) que ocupam 32 *bytes* cada uma. O servidor utiliza também o armazenamento não volátil da VM para alojar a chave pública do cliente.

O SAA e o SAABC guardam localmente as mesmas chaves criptográficas mas possuem mais ficheiros de sessão para lidar com a utilização dos *buckets* do serviço S3. A base de dados e os ficheiros SST com as entradas dos índices passam a estar armazenados no serviço S3 nos dois sistemas, ficando ainda na máquina cliente do SAABC alguns ficheiros adicionais necessários ao funcionamento dos índices (RocksDB-Cloud).

Tabela 6.7: Volume ocupado pelas estruturas do Sistema Apenas Armazenamento (SAA) para o conjunto "A" (Valores em MB).

Conjunto	Mapa de Contadores	Índice(Cli)	Índice(S3)	Base de Dados	Sessão
A	1.94	0.0	23.88	63.7	0.0048

Tabela 6.8: Volume ocupado pelas estruturas do Sistema Apenas Armazenamento com *Buffer* e *Cache* (SAABC) para cada conjunto testado, antes da operação de *flush* e sem remoções (Valores em MB).

Conjunto	Mapa de Contadores	Índice(Cli)	Índice(S3)	Base de Dados	Sessão
A	1.94	45.9	0.00008	63.7	0.0048
B	2.85	92.0	0.00008	128.4	0.0095
C	4.26	183.0	0.00008	256.1	0.0191
D	6.57	368.3	0.00008	512.0	0.0382

Tabela 6.9: Alteração do volume ocupado pelo índice no Sistema Apenas Armazenamento com *Buffer* e *Cache* (SAABC) para cada conjunto testado, após a operação de *flush* do RocksDB-Cloud (Valores em MB).

Conjunto	Índice(Cli)	Índice(S3)
A	0.217	32.0
B	0.233	64.0
C	0.748	127.0
D	1.234	254.8

O volume das estruturas do SAA, após a inserção do conjunto "A", apresenta-se na tabela 6.7. Os valores ocupados pelas estruturas no SAABC são apresentados na tabela 6.8. Nesta tabela estão os valores antes do SGBD RocksDB-Cloud ter realizado a operação de criação dos ficheiros SST. Aquando da operação de criação dos ficheiros SST, os valores de armazenamento ocupados pelo índice são alterados, como apresentado na tabela 6.9.

6.1.5 Discussão

As operações de inserção e remoção realizam a colocação das entradas nos respetivos índices da mesma forma pelo que, para o mesmo sistema e para o mesmo número de entradas, os tempos obtidos são semelhantes. A diferença no tempo total de execução deve-se às diferentes sub-operações que cada uma realiza. A operação de inserção tem de efetuar a extração de palavras-chave, atribuição do nome cifrado, registo dos nomes original e cifrado em estruturas de dados, cifra do ficheiro e por fim envio para armazenamento. A operação de remoção tem de efetuar também a verificação das palavras-chave do documento a remover, garantindo a consistência entre os dois índices, mas apenas tem

de enviar uma mensagem com o nome cifrado do ficheiro a retirar do armazenamento, que por ter menor volume de dados é mais rápido do que o envio do próprio ficheiro.

Comparando o tempo de execução total do SCA e do SAABC verifica-se que o segundo demora, em média, aproximadamente 20% mais tempo que o primeiro a realizar a operação de inserção e 13% na operação de remoção. O mesmo acontece nas operações de pesquisa, onde o SAABC tem tempos de execução total três a quatro vezes mais elevados que o SCA, aumentando a diferença após a realização das remoções.

Em relação ao SAA, os tempos de execução das operações de inserção e remoção são superiores aos dois outros sistemas testados, tendo uma duração que não é adequada a um sistema real. Nas pesquisas, as medições efetuadas mostram tempos mais aproximados aos obtidos nos dois outros sistemas. No entanto, para a mesma pesquisa, demora mais tempo que o SCA e é mais rápido que o SAABC.

Para que a operação de pesquisa devolva resultados satisfatórios, o número de palavras a selecionar dos documentos deve ser uma amostra realizada adequadamente. Quando os ficheiros a inserir contêm grandes quantidades de texto, o número de entradas a colocar no índice torna-se considerável. Isto é visível nos conjuntos de ficheiros criados e utilizados para desenvolvimento e teste dos sistemas, onde ficheiros de dimensão pequena continham números elevados de palavras-chave. Percebeu-se que este elevado número de palavras-chave criava um grande volume de entradas no índice e que era um ponto de contenção das operações. Para lidar com grandes números de palavras nos documentos pode adequar-se a extração de palavras-chave às necessidades de pesquisa particulares do utilizador. Essa adaptação é feita modificando o elemento que faz a extração das palavras-chave pois, quanto mais adequado às necessidades de pesquisa, menos entradas desnecessárias no índice de inserções existirão.

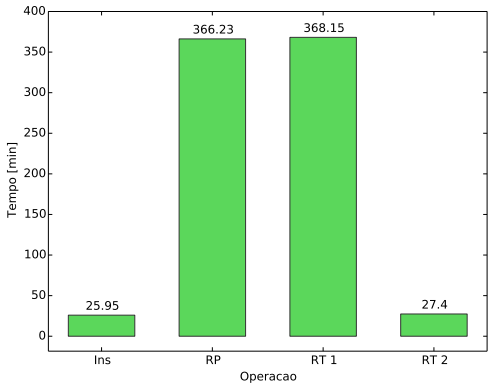
Em relação aos requisitos de armazenamento exigidos às máquinas que serão utilizadas como cliente, os valores obtidos nos testes realizados estão ao alcance de dispositivos com menores capacidades. Para o maior conjunto testado no SCA, mesmo no pior caso das remoções, os dois mapas de contadores juntos ocupariam menos de 550 MB. No sistema SAABC, o cliente teria de armazenar menos de 800 MB e este valor seria reduzido assim que o SGBD realizasse a operação de *flush*.

6.2 Necessidades da Reindexação e Libertação de Espaço no Índice (T3)

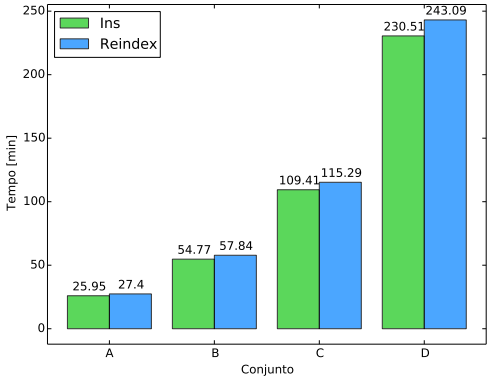
Para que seja possível reclamar espaço ocupado pelo índice de remoções, trocar as chaves criptográficas ou esconder padrões de acesso às entradas dos índices, é necessário executar uma operação de reindexação.

Na figura 6.1a são apresentados os tempos de execução de todas as operações de reindexação implementadas no SCA, onde *Ins* indica o tempo de inserção do conjunto no

sistema (para comparação), *RP* o tempo da reindexação parcial, *RT 1* o tempo da reindexação total(1) e *RT 2* o tempo da operação de reindexação total(2). Devido ao elevado tempo de execução das operações *RP* e *RT 1* no conjunto *A*, estas operações não foram testadas para os restantes conjuntos nem para os outros sistemas, pois considerou-se que os tempos obtidos não eram aceitáveis em sistemas reais. Na figura 6.1b são apresentados os tempos medidos na operação de reindexação total(2) do SCA para todos os conjuntos testados e o seu valor comparado com o tempo de inserção. Os resultados da execução da operação de reindexação total(2) no SAABC são os apresentados no gráfico da figura 6.2. Não se testou a operação de reindexação total(2) do sistema SAA por indisponibilidade de créditos.



(a) Operações no Conjunto *A*



(b) Reindexação Total (2) *versus* Inserção

Figura 6.1: Tempos de Execução das Operações de Reindexação do Sistema Computação com Armazenamento (SCA)

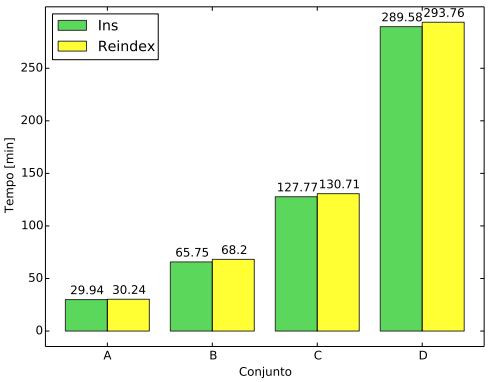


Figura 6.2: Reindexação Total (2) *versus* Inserção do Sistema Apenas Armazenamento com *Buffer* e *Cache* (SAABC).

Tabela 6.10: Variáveis utilizadas nas fórmulas dos modelos de custos.

Variável	Definição
F	Total de ficheiros cifrados a inserir/pesquisar/remover
C	Total de palavras-chave extraídas do documento
T_x	Tamanho do elemento x em GB
B	Total de *buckets* no serviço S3
x_i	Valor no mapa de contadores de inserções da palavra i
y_i	Valor no mapa de contadores de remoções da palavra i
P	Soma dos valores dos mapas de contadores $(x_i + y_i)$
Q	Diferença dos valores dos mapas de contadores $(x_i - y_i)$
S	Total de ficheiros SST
M	Total de leituras não encontradas na cache de blocos
pc	Palavra-chave retirada de um documento
doc	Representa o nome cifrado de um documento

6.2.1 Discussão

As operações de reindexação implementadas permitiram comparar as diferenças de performance das duas soluções encontradas para identificar as entradas do índice que não foram eliminadas do sistema: a pesquisa nos índices de todas as palavras-chave que estão em, pelo menos, um documento no sistema ou um novo processamento dos ficheiros para obter as palavras-chave. As pesquisas sobre o índice revelam-se muito mais lentas que o processamento dos ficheiros no cliente, como demonstrado no teste feito no SCA ao conjunto A, pelo que se considera esta abordagem inaceitável para um sistema real.

As operações de inserção e de reindexação total(2) são iguais no que respeita à inserção de entradas nos índices, pelo que os tempos obtidos nesta sub-operação são semelhantes. A diferença de tempos totais de execução é justificada pela diferença que existe nas sub-operações executadas. A operação de reindexação total (2) é a que executa mais sub-operações sobre os ficheiros pois decifra e extrai todas as palavras-chave e volta a cifrar e a enviar para armazenamento.

6.3 Modelos de Custos (T2)

Nesta secção é feita uma descrição e análise dos modelos de custos que serão imputados aos sistemas. A tabela 6.10 apresenta uma descrição das variáveis utilizadas.

6.3.1 Modelo do Sistema "Computação com Armazenamento"

O custo mensal do sistema que implementa a arquitetura "computação com armazenamento" é dado pela expressão 6.1. Esta expressão é composta por três termos que englobam o custo dos recursos necessários e o custo da realização das operações implementadas. O termo CC (Custos de Computação) designa os custos cobrados pelos recursos de computação, o termo CA (Custos de Armazenamento) designa os custos cobrados pelos recursos de armazenamento não volátil e o termo $CCom$ (Custos de Comunicação) designa os custos cobrados pelo volume de dados transferidos entre o cliente e o servidor.

$$Total_{mensal} = CC + CA + CCom \tag{6.1}$$

6.3.1.1 Custos de Computação

Esta variável indica o valor a cobrar mensalmente pela utilização da máquina virtual onde executa o servidor do sistema, que opera sobre o índice e sobre a base de dados. Como o sistema apenas prevê a utilização de um servidor, o custo é dado pela expressão $Custo_{hora} \times N^{\underline{o}} horas$. No caso do fornecedor de serviços AWS, e para o seu serviço EC2, o número de horas mensal é calculado dividindo o número de horas existente num ano pelo número de meses ($(24 \times 365) \div 12 = 730\, horas/mês$). O custo da VM por hora irá depender dos recursos escolhidos.

6.3.1.2 Custos de Armazenamento

O custo da memória não volátil é calculado pela expressão 6.2, onde QA representa a quantidade de armazenamento pretendida e $Custo_{GB}$ o custo cobrado pelo fornecedor por GB/mês. No caso de serem utilizadas mais instâncias, estes valores teriam de ser multiplicados pelo número de instâncias pretendidas. É necessário considerar que o armazenamento tem de ter capacidade para o índice de inserções (II), indice de remoções (IR) e para a base de dados.

$$CA = (QA_{II} + QA_{IR} + QA_{BDC}) \times Custo_{GB} \tag{6.2}$$

6.3.1.3 Custos de Comunicação

No que respeita aos custos imputados à comunicação entre o cliente e o servidor, estes dividem-se pelas várias operações que o sistema oferece aos utilizadores. Os $CCom$ resultarão do somatório dos custos das operações de inserção, pesquisa, remoção e reindexação realizadas durante o mês. Neste modelo de custos não se contabilizou as mensages de *setup* que são enviadas sempre que o cliente inicia a utilização de um novo índice pois estas mensagens são apenas enviadas na primeira vez que um cliente se liga ao servidor ou na realização de uma operação de reindexação. Como a sua dimensão é de 32 *bytes*, a sua contribuição para os custos do sistema podem-se considerar pouco relevantes.

Os custos de inserção são dados pela expressão 6.3, que contabiliza o tráfego do cliente para o servidor.

$$\sum_{i=1}^{F}\left[\sum_{j=1}^{C}\left((UT_j, doc_i)\times T_{par}\right)+T_i\right]\times Custo_{GB} \tag{6.3}$$

No caso das pesquisas, a expressão é dada pela fórmula 6.4. Nesta operação é necessário ter em conta que a primeira parcela da soma refere-se a tráfego que é enviado do cliente para o servidor e a segunda parcela a tráfego do servidor para o cliente, que corresponde ao envio dos ficheiros encontrados.

$$\left[T_{msg\,pesquisa}\times C\times Custo_{GB}\right]+\left[\sum_{i=1}^{F}(T_i)\times Custo_{GB}\right] \tag{6.4}$$

O custo das operações de remoção é dado pela expressão 6.5, que calcula o custo do tráfego enviado do cliente para o servidor.

$$\left[\sum_{i=1}^{F}\left[\sum_{j=1}^{C}\left((UT_j, doc_i)\times T_{par}\right)\right]+(i\times T_{msg\,del})\right]\times Custo_{GB} \tag{6.5}$$

Por último, os custos da operação de reindexação são dados pela expressão 6.6. A primeira parcela da soma diz respeito ao tráfego de dados do servidor para o cliente, na sequência do pedido de realização da operação de reindexação, onde o servidor envia todos os ficheiros cifrados armazenados para o cliente. A segunda parcela da soma diz respeito ao envio dos novos ficheiros cifrados e novas entradas do índice de inserções do cliente para armazenamento no servidor, representado pela expressão 6.3

$$\left[\sum_{i=1}^{F}(T_i)\times Custo_{GB}\right]+Custo_{ins} \tag{6.6}$$

6.3.2 Modelo do Sistema "Apenas Armazenamento"

O custo mensal do sistema que implementa a arquitetura "apenas armazenamento" é dado pela expressão 6.7. Esta expressão é composta por três termos, nomeadamente o termo COp (Custos das Operações) que designa os custos cobrados pelas operações solicitadas ao serviço, o termo CA (Custos de Armazenamento) que designa os custos cobrados pelos recursos de armazenamento não volátil e o termo $CCom$ (Custos de Comunicação) que designa os custos cobrados pelo volume de dados transferidos entre o cliente e o serviço de armazenamento.

$$Total_{mensal} = COp + CA + CCom \tag{6.7}$$

6.3.2.1 Custos das Operações

Como todas as computações são realizadas na máquina cliente, os gastos com a VM são substituídos pelos custos das operações solicitadas ao serviço de armazenamento. Para

as operações de inserção de ficheiros, o custo é dado pela expressão 6.8. Nesta expressão são contabilizados o custo das operações de inserção de pares (UT, nome cifrado) no serviço de alojamento, que materializam as entradas do índice de inserções, e os pedidos de inserção de ficheiros.

$$\sum_{i=1}^{F}\left[\sum_{j=1}^{C}(UT_j, doc_i)\times Custo_{POST/PUT}\right]+(F\times Custo_{POST/PUT}) \tag{6.8}$$

No caso em que o sistema utilize o SGBD RocksDB-Cloud, os custos da operação de inserção são dados pela expressão 6.9. Os pedidos para inserção dos ficheiros SST estão dependentes das configurações de tamanho da *memtable* e dos ficheiros SST, uma vez que sempre que a primeira fica totalmente preenchida gera ficheiros SST e estes são enviados para o serviço S3.

$$[S\times Custo_{POST/PUT}]+[F\times Custo_{POST/PUT}] \tag{6.9}$$

Os custos das pesquisas dependem da forma como a implementação trata as remoções, uma vez que ao utilizar dois índices serão feitos mais pedidos ao serviço de alojamento. Concluiu-se que o número de operações `GET` solicitadas numa pesquisa no SAA, para identificação dos ficheiros onde a palavra se encontra, é dado pela expressão $P_i = x_i + y_i$, onde x e y representam o valor do contador de inserções e o valor do contador de remoções respetivamente, aos quais é somado uma unidade (uma vez que o contador inicia em zero, ou zero caso a palavra não esteja presente no mapa de contadores). Uma vez encontrado o valor P pode-se determinar o custo total da pesquisa, que é dado pela expressão 6.10. Esta expressão calcula o custo de pesquisa de uma única palavra.

$$[P\times Custo_{GET}]+[F\times Custo_{GET}] \tag{6.10}$$

No SAABC o número de solicitações ao serviço de armazenamento poderá ser menor em virtude da existência da *block cache*. Os custos da pesquisa de uma palavra são dados pela expressão 6.11, onde só serão solicitados os blocos dos ficheiros SST que não estão na *cache* na máquina cliente e aos quais se somam os pedidos que solicitam os ficheiros.

$$[M\times Custo_{GET}]+[F\times Custo_{GET}] \tag{6.11}$$

O custo das remoções no SAA compreende as inserções dos dados que correspondem ao índice de remoções e os pedidos para eliminação física do ficheiro do serviço de alojamento. Estes custos são dados pela fórmula 6.12.

$$\sum_{i=1}^{F}\left[\sum_{j=1}^{C}(UT_j, doc_i)\times Custo_{POST/PUT}\right]+(F\times Custo_{DEL}) \tag{6.12}$$

As remoções no SAABC são dadas pela expressão 6.13.

$$[S \times Custo_{POST/PUT}] + [F \times Custo_{DEL}] \tag{6.13}$$

Por fim, os custos da operação de reindexação total(2) são dados pela expressão 6.14. O cálculo desta operação é composta pelo custo dos pedidos de todos os ficheiros que estão armazenados e pelo custo de inserção de todos os ficheiros e novas entradas no serviço de armazenamento. A expressão é a mesma para os dois sistemas, diferindo no termo $Custo_{ins}$, que para o primeiro caso representa a expressão 6.8 e para o segundo a expressão 6.9.

$$[F \times Custo_{GET}] + [B \times Custo_{DEL}] + Custos_{ins} \tag{6.14}$$

6.3.2.2 Custos de Armazenamento

Os custos de armazenamento são calculados utilizando a expressão 6.2, uma vez que o tipo de dados colocados em armazenamento no fornecedor de serviços são os mesmos.

6.3.2.3 Custos de Comunicação

Os custos de comunicação da operação de inserção do SAA são também calculados com a expressão 6.3, uma vez que o tipo e quantidade de dados que são enviados pelo cliente são iguais. Para o SAABC, estes custos serão calculados pela expressão 6.15.

$$\left[\sum_{i=1}^{S}(T_i) + \sum_{j=1}^{F}(T_j)\right] \times Custo_{GB} \tag{6.15}$$

No que respeita às pesquisas no SAA, o custo é dado pelas expressões 6.16 e 6.17. A primeira expressão calcula o custo de pesquisa das entradas que pertencem ao índice, estando a comunicação do cliente para o servidor no primeiro termo da soma e a comunicação do servidor para o cliente no segundo termo. É utilizado o valor P calculado para a expressão 6.10. A segunda expressão calcula o custo da obtenção dos ficheiros identificados e que contêm a palavra, estando no primeiro termo da soma os pedidos feitos do cliente ao servidor e no segundo termo os ficheiros enviados do servidor para o cliente.

$$\sum_{i=1}^{C}[((P_i \times T_{UT}) \times Custo_{GB}) + ((P_i \times T_e) \times Custo_{GB})] \tag{6.16}$$

$$\sum_{i=1}^{F}[(Q_i \times T_{index} \times Custo_{GB}) + (Q_i \times T_{doc} \times Custo_{GB})] \tag{6.17}$$

As pesquisas no SAABC são calculadas com a expressão 6.18, que engloba os custos de solicitar os blocos dos ficheiros SST que não existem na *cache* no cliente. A esta expressão terão de ser somados os custos de solicitar os ficheiros encontrados na pesquisa, dados pela expressão 6.17.

$$(M \times T_{Bloco} \times Custo_{GB}) \tag{6.18}$$

Os custos de comunicação das operações de remoção no SAA são calculados de acordo com a expressão 6.5, já utilizada na arquitetura "computação com armazenamento". No SAABC os custos da remoção serão calculados de forma diferente, utilizando a expressão 6.19. A primeira parte calcula o custo das remoções lógicas (ou inserções no índice de remoções) e a segunda calcula os custos de ordenar a remoção dos ficheiros do serviço de armazenamento. Nesta operação todas as comunicações são pedidos do cliente para o servidor.

$$\sum_{i=1}^{S} (T_i \times Custo_{GB}) + (F \times T_{msg\,del} \times Custo_{GB}) \tag{6.19}$$

A última operação considerada é a reindexação total(2) e o custo de comunicação desta operação é dado pela soma da expressão 6.6 com a expressão do $Custo_{ins}$, que será a expressão 6.3 se utilizarmos o SAA, ou à expressão 6.15 se utilizarmos o SAABC.

6.3.3 Comparação de Custos para o Conjunto "A"

Uma vez descritos os modelos de custos de cada um dos sistemas, faremos uma comparação de custos de utilização mensal para o conjunto A.

6.3.3.1 Exemplo para o Sistema "Computação com Armazenamento"

O sistema foi testado com uma instância *t2.medium* localizada na região *US East (Northern Virginia)*. Esta instância tem um custo mensal de \$33.87 USD ($CC = 0.0464 \times 730$), de acordo com os dados disponibilizados em [50].

No que respeita aos custos de armazenamento CA, esta instância disponibiliza armazenamento não volátil no serviço EBS, que possibilita reduzir e aumentar a capacidade de acordo com as necessidades. Foram adjudicados 40 GB de armazenamento e o valor a pagar mensalmente é de $CA = 40 \times 0.10$, ou seja \$4.00 USD como indicado na página [50]. Apenas com estes elementos, a manutenção da VM custa \$37.87 USD mensais.

Os custos de comunicação cobram pela quantidade de dados transferidos do servidor para o cliente e as operações que podem implicar grandes volumes de dados são as pesquisas, dependendo do número de resultados que são obtidos, e as operações de reindexação. Numa operação de pesquisa, o número de resultados devolvidos é variável assim como o tamanho dos ficheiros. Na operação de reindexação, e considerando apenas a reindexação total(2), todos os ficheiros cifrados são enviados do servidor para o cliente. É possível fazer uma previsão dos custos em função do tamanho médio dos ficheiros pois o fornecedor de serviços AWS cobra \$0.09 USD por cada GB transferido para a máquina cliente.

Considerando o conjunto A e que num mês seriam pesquisadas 50000 palavras-chave, sendo essas palavras as apresentadas no ponto 6.1.2 (10000 pesquisas para cada palavra), seriam obtidos 3100000 ficheiros. O custo das inserções seria $Custo_{ins} = [(995171 \times 24\,bytes + 62 \times 1MB) \times 0)]$. No que respeita às pesquisas, o custo seria $Custo_{pesq} = (600 \times 50000 \times 0) + (3100000 \times 1MB \times 0.09)$ que totalizaria o valor de \$279 USD. Por fim, o custo da reindexação seria $Custo_{reindex} = [(62 \times 0.001) \times 0.09] + [(62 \times 0.001) + (995171 \times 24\,bytes)] \times 0$, o que totalizaria \$0.00558 USD. O custo mensal deste conjunto de operações totaliza $CCom = 279.01USD$.

Somando todos os valores obtidos temos $Total_{mensal} = \$367.5$ USD.

6.3.3.2 Exemplo para Sistema "Apenas Armazenamento"

Para efeitos de comparação, foi calculado o custo mensal para a mesma quantidade de armazenamento considerada para o SCA. Assim, se o armazenamento total for de 40 GB no serviço S3, o custo mensal será $CA = 40 \times 0.023$, ou seja \$0.92 USD por mês.

Para calculo dos custos das operações, na inserção temos de considerar todos os pedidos realizados ao sistema para inserir o conjunto. Esse custo é de $((995171 \times 0.005) \div 1000) + ((62 \times 0.005) \div 1000) = \$4.97617USD$. Para calcular o custo das operações das pesquisas é necessário contabilizar as entradas que serão pesquisadas no índice de inserções e no índice de remoções. O número de pedidos ao serviço é dado pela soma dos valores dos contadores de cada palavra. Para realizar as mesmas 50000 pesquisas seriam feitos 3100000 pedidos ao serviço de alojamento S3 apenas para verificar as entradas no índice de inserções (uma vez que nenhuma remoção foi feita), que teriam um custo de $(3100000 \times 0.0004) \div 1000 = \$1.24USD$. A este valor é necessário somar os pedidos para solicitar os ficheiros encontrados, que teria um custo de $(3100000 \times 0.0004) \div 1000 = \$1.24USD$ (os valores são iguais pois não existem remoções). As pesquisas teriam um valor total de \$2.48 USD. Por último, o custo das operações da reindexação é dado pelo custo dos pedidos de transferência dos ficheiros para o cliente, ou seja, $(62 \times 0.0004) \div 1000$ somados aos custos de inserção, o que totaliza \$4.9762 USD. O termo COp totaliza \$12.428 USD.

Para o cálculo dos custos de comunicação, teríamos um custo de inserção do conjunto de \$0 USD, uma vez que o serviço AWS não cobra o envio de dados para o serviço. Os custos das pesquisas têm de ter em conta os pedidos feitos às entradas do índice, sendo dados pela expressão $(3100000\,pedidos \times 16\,bytes \times \$0USD) + (8\,bytes \times 3100000 \times \$0.09\,USD) + (8\,bytes \times 3100000 \times \$0USD) + (3100000 \times 1\,MB \times \$0.09\,USD) = \$279USD$. A reindexação será dada pelos custos de transferência de todos os ficheiros armazenados novamente para o cliente e somados aos custos de inserção, ou seja $(62\,ficheiros \times 1\,MB \times \$0.09USD) + \$0\,USD = \$0.00558USD$. Os custos de comunicação totalizam \$279.01 USD por mês.

Somando os valores obtidos no exemplo, obtém-se um valor mensal de \$292.4 USD.

6.3.3.3 Exemplo para Sistema "Apenas Armazenamento" com *Buffer* e *Cache*

Para esta solução, o custo de armazenamento é igual ao calculado no ponto 6.3.3.2.

Para o cálculo dos custos das operações e das comunicações é necessário considerar os tamanhos definidos para as *memtables* e ficheiros SST do RocksDB-Cloud. Utilizando os definidos na implementação do Sophos, temos 1 GB de tamanho das *memtables* e ficheiros SST de 192 MB. Para o conjunto A, as entradas do índice de inserções e do índice de remoções totalizam $\approx$ 24 MB, pelo que a *memtable* não será completamente preenchida.

Os custos das operações para a inserção serão dados pela expressão $(1\,SST \times 0.005) \div 1000$, que calcula o custo de envio do ficheiro SST para o armazenamento no serviço S3, ao qual será somado o custo das operações de enviar os ficheiros para armazenamento, dados pela expressão $(62 \times 0.005) \div 1000$, que totaliza \$0.000315 USD. As pesquisas são calculadas pela expressão $((3100000 \times 0.0004) \div 1000) = \$1.24$, que indica o custo de solicitar todos os blocos do índice no pior caso, em que todos os blocos não estão na *block cache*, e ao qual é somado o valor dos pedidos que solicitam os ficheiros ao serviço de armazenamento, dado pela expressão $((3100000 \times 0.0004) \div 1000) = \$1.24$, totalizando \$2.48 USD. A reindexação será dada pela expressão $(62 \times 0.0004) \div 1000$, que contabiliza os pedidos a fazer para obter no cliente todos os ficheiros armazenados, ao qual será somado o valor de inserção, totalizando \$0.00034 USD. O custo total das operações é de \$2.48 USD.

Por último, os custos da comunicação. A inserção é calculada com a expressão $0.192\,GB \times \$0\,USD$, assumindo que o ficheiro SST é enviado completo (apesar da dimensão do índice não preencher um ficheiro SST completo), somando o valor do envio dos ficheiros cifrados, dado pela expressão $(62 \times 0.001 \times \$0 USD)$. Os custos de comunicação das pesquisas são compostos pelos custos da transferência dos blocos de 4 KB que compõem os ficheiros SST que não estejam na *block cache*. No pior caso, todas as entradas a solicitar ao índice estão em blocos diferentes, o que significa que serão solicitados tantos blocos quantos os documentos onde a palavra existe, dados pela expressão $(3100000 \times 0.000004 \times \$0.09\,USD)$ que totaliza \$1.116 USD, ao qual será somado o custo do pedido dos ficheiros e da transferência de ficheiros, dado pela expressão $(8\,bytes \times 3100000 \times \$0 USD) + (3100000 \times 0.001\,GB \times \$0.09\,USD) = \$279\,USD$. Por fim, a operação de reindexação teria um custo de $(62\,ficheiros \times 0.001 \times \$0.09 USD) + \$0\,USD = \$0.00558 USD$. Os custos de comunicação totalizam \$280.116 USD por mês.

Esta sistema, para o exemplo dado, tem um custo mensal de \$283.516 USD.

6.3.3.4 Comparação de Custos dos Conjuntos Testados

O exemplo apresentado para o conjunto A foi utilizado para obter os custos para os restantes conjuntos. Os valores obtidos estão na tabela 6.11, onde se apresenta a origem da despesa e o conjunto que lhe corresponde. É possível verificar que os custos do SAABC são os mais baixos apesar de apresentar os custos mais elevados para as pesquisas. No entanto, os valores apresentados para a operação de pesquisa neste sistema assumem que

Tabela 6.11: Comparação dos custos de operação mensais por conjunto.

Sistema	Despesa	A	B	C	D
SCA	VM	33.87	33.87	33.87	33.87
	Armazenamento	4.00	4.00	4.00	4.00
	Inserção	0	0	0	0
	Pesquisa	279.0	562.5	1120.5	2236.5
	Reindexação	0.00558	0.01125	0.02241	0.04473
	Total	316.88	600.38	1158.39	2274.41
SAA	VM	N/A	N/A	N/A	N/A
	Armazenamento	0.92	0.92	0.92	0.92
	Inserção	4.98	9.98	19.83	39.90
	Pesquisa	281.48	567.51	1130.47	2256.40
	Reindexação	4.98	9.99	19.85	39.95
	Total	292.36	588.40	1171.07	2337.17
SAABC	VM	N/A	N/A	N/A	N/A
	Armazenamento	0.92	0.92	0.92	0.92
	Inserção	0.00032	0.00063	0.00125	0.0249
	Pesquisa	282.60	569.75	1134.94	2265.33
	Reindexação	0.0059	0.0119	0.0238	0.0474
	Total	283.52	570.68	1135.89	2266.30

nenhum bloco existe na *block cache*, o que não se verificará numa utilização real, pelo que estes custos poderão ser menores.

6.3.4 Discussão

Através das previsões de custos feitas para a simulação de operações no conjunto *A*, apresentadas nos pontos 6.3.3.1, 6.3.3.2 e 6.3.3.3, foi possível perceber que existem dois aspetos distintos que devem ser analisados separadamente: por um lado os custos de inserção dos elementos no sistema e os valores cobrados para manter o sistema em funcionamento e, por outro, os custos das pesquisas.

No que respeita aos custos de inserção, o SCA não cobra nenhum valor pelo envio dos ficheiros e das entradas dos índices. Os restantes sistemas apresentam custos relacionados com as operações solicitadas e pelo volume de dados enviados para o serviço de armazenamento. No entanto, os custos de manutenção da VM no SCA superam várias vezes o custo de armazenamento do serviço S3.

Nas pesquisas, o SCA apresenta custos mais baixos mas, em virtude do valor cobrado para manter a VM em funcionamento, o custo final é mais elevado que o dos restantes sistemas para os conjuntos *A* e *B*. Esta diferença altera-se nos conjuntos *C* e *D*, em que o SAA passa a ter custos mais elevados, por causa dos valores mais elevados cobrados na inserção e pesquisa face a um custo da VM que se mantém constante no SCA.

É possivel afirmar que o SAABC apresenta vantagem em termos dos custos, em virtude

da realização de todas as computações na máquina cliente e da forma como envia as entradas para os índices.

6.4 Avaliação de Segurança dos Sistemas

Com o desenvolvimento de novas funcionalidades é necessário verificar se as informações reveladas ao servidor se mantêm iguais às garantidas pelo esquema de CP utilizado. Como referido no documento do Sophos [2], ao modificar-se o esquema para suportar remoções a função de *leakage* permanece igual mas passa a ser revelado ao servidor o tipo de operação que é realizada (inserção ou remoção). No entanto, o objetivo do sistema é o tratamento de ficheiros e ao associar isso às operações sobre o índice, o sistema pode revelar mais informação acerca das operações realizadas.

Os esquemas de CP caracterizam a informação revelada ao servidor através de uma função de leakage $\mathcal{L}$ que é definida pelo leakage de cada uma das operações suportadas. No caso dos sistemas desenvolvidos, o leakage será definido pela função $\mathcal{L} = (\mathcal{L}_{Setup}, \mathcal{L}_{Ins}, \mathcal{L}_{Rem}, \mathcal{L}_{Pesq}, \mathcal{L}_{Reindex})$. A função de inicialização do sistema não revela informações ao fornecedor de serviços, pelo que pode ser definida pela função $\mathcal{L}_{Setup} = \perp$. As restantes funções de *leakage* são definidas nas secções seguintes.

6.4.1 *Leakage* nas operações de inserção

Os sistemas desenvolvidos iniciam a inserção de ficheiros extraíndo as palavras-chave e atribuíndo o nome de armazenamento ao ficheiro. Após este tratamento inicial, o ficheiro é cifrado e enviado para armazenamento. Por último, são enviadas as mensagens que permitem inserir as entradas no índice de inserções.

Nos três sistemas implementados o envio dos ficheiros cifrados para alojamento revela informações ao fornecedor de serviços, nomeadamente o número de ficheiros cifrados enviados, os respetivos nomes de armazenamento, o tamanho de cada um dos ficheiros, a data/hora de inserção e o tamanho total do conjunto.

Posteriormente, a colocação de entradas no índice de inserções será diferente de acordo com o sistema considerado. O SCA e SAA enviam todos os pares (UT, nome cifrado) para o índice logo após o envio dos ficheiros cifrados. Quando esta operação é realizada com mais que um ficheiro em simultâneo, o fornecedor de serviços consegue saber que está a ser realizada uma operação de inserção, o número total de entradas que estão a ser colocadas no índice de inserções, a data/hora de inserção de cada uma das entradas mas não consegue estabelecer uma relação entre os vários ficheiros inseridos e as entradas enviadas para o índice. No entanto, se apenas um ficheiro for inserido, a associação entre o ficheiro cifrado enviado e as entradas colocadas no índice é revelada. Isto significa que, de cada vez que um ficheiro for inserido de forma isolada, o fornecedor de serviços consegue associar o ficheiro ao conjunto de entradas do índice que lhe correspondem. O SAABC é o que menos informação revela ao fornecedor de serviços. A introdução das entradas

no índice é feita inicialmente na máquina cliente e só após a passagem dos elementos da *memtable* para os ficheiros SST é que estes serão armazenados no serviço S3, permitindo um desfasamento temporal entre as inserções de ficheiros na base de dados e as inserções de entradas nos índices. Esta forma de funcionamento da *memtable* permite que funcione como um *buffer* pesquisável, sendo uma solução para o problema da inserção de um único ficheiro pois o SGBD RocksDB-Cloud permite configurar o sistema para que não realize a operação de *flush* em nenhum ocasião excepto quando a *memtable* ficar completamente ocupada. O SCA apesar de utilizar o SGBD RocksDB para implementar os índices e de ter disponível esta configuração, não pode utilizar esta solução uma vez que os índices se localizam no servidor e este tem acesso às entradas do índice e aos ficheiros cifrados antes destes serem colocados na *memtable* e no disco local da VM.

Apesar do aumento da informação revelada ao introduzir a manipulação de ficheiros, todos os sistemas continuam a garantir a propriedade de *forward privacy* pois o fornecedor de serviços, aquando da inserção de um novo documento, não consegue saber que este contém palavras pesquisadas anteriormente uma vez essa propriedade é garantida pelas construções criptográficas do Sophos.

Desta forma, pode definir-se a função de *leakage* como $\mathcal{L}_{Ins} = (\mathcal{L}_{IF}, \mathcal{L}_{IE})$, em que o *leakage* da inserção do sistema é o resultado do *leakage* da inserção dos ficheiros ($\mathcal{L}_{IF}$) e da inserção das entradas no índice ($\mathcal{L}_{IE}$). O *leakage* da inserção de ficheiros é dado pela função:

$$\mathcal{L}_{IF}(\mathbf{D}) = (|f_i|, t_f, |\mathbf{F}|, N)$$

onde $\mathbf{D}$ representa o conjunto de ficheiros em claro a introduzir no sistema, $|f_i|$ o tamanho de cada um dos ficheiros cifrados enviados, t_f a data/hora da inserção de cada um dos ficheiros, $|\mathbf{F}|$ o volume total do conjunto de ficheiros cifrados e N o número de ficheiros no conjunto.

O *leakage* da inserção de entradas no índice do SCA e do SAA é dado pela função:

$$\mathcal{L}_{IE}^{SCA/SAA}(ins, w, n) = (ins, t_{ins}, \mathbf{F})$$

onde a colocação de um novo par (palavra-chave (w), nome do documento (n)) no sistema revela a operação (ins) que se está a realizar, pois o servidor sabe qual o índice que é utilizado para as inserções, a data/hora da colocação da entrada no índice e qual o conjunto de ficheiros cifrados $\mathbf{F}$ que lhe corresponde. No SAABC o *leakage* da colocação de entradas no índice de inserções será diferente devido à utilização da *memtable* como *buffer* e é dado pela função:

$$\mathcal{L}_{IE}^{SAABC}(\mathbf{I}) = (ins, t_{ins}, T)$$

onde $\mathbf{I}$ representa o conjunto de entradas do índice que estão na *memtable*, ins a operação de inserção, T o total de entradas do índice contidos nos ficheiros SST e t_{ins} a data/hora de colocação desses ficheiros no serviço de armazenamento.

6.4.2 *Leakage* nas operações de remoção

As operações de remoção de ficheiros completos podem revelar as entradas que pertencem a um ficheiro em particular uma vez que a sequência de operações é igual à inserção de ficheiros, mudando apenas o índice onde são colocadas as entradas das remoções lógicas (índice de remoções). No SCA e no SAA, se apenas for realizada a remoção de um documento, o fornecedor do serviço consegue associar as entradas enviadas para o índice de remoções ao ficheiro indicado para remoção. O SAABC consegue evitar esta situação como descrito na operação de inserção. Se for identificado mais do que um documento para remoção, esta associação entre documentos e entradas no índice não é possível em nenhum dos três sistemas.

O *leakage* da operação de remoção é também composto por duas funções de *leakage* distintas $\mathcal{L}_{Rem} = (\mathcal{L}_{RF}, \mathcal{L}_{RL})$, nomeadamente o *leakage* da remoção dos ficheiros ($\mathcal{L}_{RF}$) e da inserção das entradas no índice de remoções, ou remoções lógicas ($\mathcal{L}_{RL}$). O *leakage* da remoção de ficheiros é igual para os três sistemas implementados e é dado pela função:

$$\mathcal{L}_{RF}(\mathbf{S}) = (t_r\,, |\mathbf{S}|, S)$$

onde $\mathbf{S}$ representa o conjunto de nomes de armazenamento dos ficheiros que se pretende eliminar, t_r a data/hora da remoção do ficheiro, $|\mathbf{S}|$ o volume de ficheiros indicados para remoção e S o número de ficheiros removidos.

À semelhança do que acontece na operação de inserção, a função de *leakage* das remoções lógicas para o SCA e para o SAA é dada pela expressão:

$$\mathcal{L}_{RL}^{SCA/SAA}(rem, w, n) = (rem, t_{rem}, \mathbf{S})$$

onde é revelado a operação que está a ser executada (rem), a data/hora da colocação das entradas no índice de remoções e o conjunto de ficheiros removidos a que as entradas do índice de remoções dizem respeito. No SAABC o *leakage* da colocação de entradas no índice de remoções é dado pela função:

$$\mathcal{L}_{RL}^{SAABC}(\mathbf{R}) = (rem, t_{rem}, Q)$$

onde R representa o conjunto de entradas do índice que estão na *memtable*, rem a operação de remoção lógica, Q o total de entradas do índice contidos nos ficheiros SST e t_{rem} a data/hora de colocação desses ficheiros no serviço de armazenamento.

6.4.3 *Leakage* nas operações de pesquisa

Ao ser realizada uma pesquisa no SCA, o servidor consegue saber quais as entradas dos índices que são acedidas e se para um determinado documento a operação com essa palavra foi apenas de inserção ou se foi posteriormente removida e qual o ficheiro que lhes corresponde, pois a decifra das entradas do índice e a diferença entre as inserções e remoções são realizadas no servidor. Este *leakage* corresponde ao definido em [2].

O SAA realiza a decifra das entradas dos índices na máquina cliente, revelando apenas quais foram acedidas. Os nomes de armazenamento dos ficheiros que correspondem à pesquisa são revelados quando estes são solicitados ao serviço de armazenamento para que possam ser mostrados ao utilizador. No entanto, como este sistema possibilita a inserção e remoção individual de ficheiros, permite que o fornecedor de serviços faça uma associação direta entre as entradas colocadas no índice e o ficheiro manipulado. Este mapeamento permite descobrir, durante uma operação de pesquisa, quais os ficheiros que continham a palavra que está a ser pesquisada e que foram entretanto removidos, o que contraria a definição de *backward privacy* apresentada em [15], que estabelece que um sistema garante esta propriedade se esconder os identificadores dos ficheiros (ou nome de armazenamento) que continham a palavra que está a ser pesquisada mas que foram, entretanto, apagados. Desta forma, o SAA mantém a propriedade de *forward privacy* mas não garante a propriedade de *backward privacy*. A função de *leakage* pode ser definida pela expressão

$$\mathcal{L}^{SAA}_{Pesq}(w) = (pp(w), \mathbf{C})$$

onde são reveladas as entradas dos índices que são acedidas, ou padrão de pesquisa pp, e o conjunto de documentos $\mathbf{C}$ que estão associados a essas entradas. O conjunto $\mathbf{C}$ que contém os nomes de armazenamento dos ficheiros pretendidos só é revelado aquando a solicitação dos ficheiros ao serviço de armazenamento, para que possam ser apresentados ao cliente.

O SAABC, ao utilizar a *memtable* como um *buffer* pesquisável, a *cache* no cliente e pela forma como obtém os elementos do índice do serviço de armazenamento, revela menos informação que a definida para sistemas que garantem *backward privacy* com um *leakage* do tipo I (*Backward Privacy with Insertion Pattern*). Pela definição, os sistemas com este tipo de *leakage* revelam ao fornecedor de serviços:

1. Número e tipo da operação de atualização associada com a palavra a pesquisar;
2. Identificadores dos ficheiros que contêm a palavra pesquisada e que estão atualmente na base de dados;
3. Data e hora da inserção dos ficheiros que contêm a palavra pesquisada.

Em relação ao ponto 1, e mesmo nos casos em que nenhuma entrada esteja na *cache* de blocos localizada na máquina cliente, como os blocos que compõem os ficheiros SST juntam várias entradas do índice, o servidor não consegue descobrir quais são aquelas que estão a ser efetivamente procuradas pela operação de pesquisa ou até, se no mesmo bloco não estão várias entradas, escondendo dessa forma o número de atualizações associadas à palavra pesquisada. O fornecedor de serviços fica apenas a saber se a palavra pesquisada já foi removida de algum documento no caso de serem solicitados blocos ao *bucket* do índice de remoções. Caso todas as entradas do índice estejam ainda na *memtable* ou todas as entradas pesquisadas estejam em blocos na *cache*, o tipo de operação de atualização

associada à palavra pesquisada é também ocultado. A informação do ponto 2 também não é revelada na sequência da operação de pesquisa pois a decifra dos nomes cifrados em nomes de armazenamento é feita na máquina cliente. O fornecedor de serviços acaba por conhecer esta informação mas como resultado da solicitação dos ficheiros ao serviço de armazenamento. O mesmo acontece com o ponto 3, onde esta informação é escondida durante a operação de pesquisa dos índices cifrados, sendo revelada aquando da solicitação dos ficheiros que correspondem ao resultado da pesquisa. Todas as restantes informações, que correspondem ao *leakage* dos tipos II e III, são ocultadas do fornecedor de serviços em virtude da utilização *memtable* e controlo da operação de *flush*, permitindo ocultar o tempo real e ordem de realização das operações sobre os índices e a associação direta de entradas dos índices de inserção e remoção. Para este sistema, o *leakage* da pesquisa pode ser definido como a revelação parcial ou total dos blocos dos ficheiros SST de inserção, ou de inserção e remoção, associados à palavra pesquisada e é dado pela função

$$\mathcal{L}_{Pesq}^{SAABC}(w) = (\mathbf{B}, \mathbf{C})$$

onde **B** indica o número de blocos de ficheiros SST solicitados ao serviço de armazenamento durante a pesquisa nos índices e **C** o conjunto de nomes de armazenamento que satisfazem a pesquisa. À semelhança do SAA, esta informação só é revelada quando o cliente solicita os ficheiros ao serviço de armazenamento e não uma consequência da decifra das entradas do índice, como acontece no SCA.

6.4.4 *Leakage* nas operações de reindexação

Se a operação de reindexação utilizar a pesquisa sobre o índice, o *leakage* permanece o mesmo apresentado anteriormente para as operações de pesquisa e de inserção.

No caso da operação de reindexação que não utiliza as pesquisas sobre o índice, substituindo essa operação por uma nova operação de extração de palavras-chave no cliente, o *leakage* das pesquisas é evitado, sendo revelado ao servidor a informação revelada nas operações de inserção.

6.4.5 Discussão

A análise efetuada permitiu perceber que ao introduzir a manipulação de ficheiros reais é possível revelar mais informação ao servidor ou fornecedor de serviços, permitindo que este associe entradas dos índices aos ficheiros cifrados correspondentes, situação que surge no SCA e no SAA. No caso específico do SAA este *leakage* é suficiente para impedir a garantia da propriedade de backward privacy. Contudo, e apesar do aumento da informação revelada pela introdução da manipulação de ficheiros, estes dois sistemas mantêm a propriedade de *forward privacy* do esquema de CP Sophos.

O SAABC é o que garante mais segurança, devido à forma como está implementado. A forma de funcionamento do SGBD RocksDB-Cloud permite que a *memtable* funcione como um *buffer* para as entradas dos índices, impossibilitando a associação dos conjuntos

de ficheiros inseridos às entradas colocadas nos índices. A maneira como são lidos os ficheiros SST do serviço de armazenamento, em conjunto com a operação da *cache* de blocos, não permite ao fornecedor de serviços saber quais as entradas dos índices que estão a ser procuradas. Esta construção permite que o sistema garanta a propriedade de *backward privacy*, apesar do esquema utilizado apenas garantir *forward privacy*.

6.5 Notas Finais de Capítulo

Neste capítulo foram apresentados os resultados da avaliação de performance dos sistemas desenvolvidos. Apresentaram-se também os modelos de custos, permitindo comparar os gastos e perceber qual o sistema que tem menores custos de utilização. Terminou-se com a avaliação de segurança dos três sistemas, identificando-se o *leakage* introduzido pela incorporação das operações sobre ficheiros nas operações sobre os índices e analisando o *leakage* de cada sistema nas operações de pesquisa, permitindo encontrar o sistema que menos informação revela ao fornecedor de serviços. Este capítulo finaliza a descrição do trabalho desenvolvido sendo apresentadas as conclusões no próximo capítulo.

Capítulo 7

Conclusão

O desenvolvimento de um sistema de CP eficiente, seguro, fácil de utilizar e com custos adequados permitirá uma utilização mais segura e privada dos recursos de armazenamento oferecidos pelos fornecedores de serviços na *cloud*. Investigou-se a literatura mais recente sobre esquemas de CP e foram identificados e apresentados cinco tópicos que devem ser considerados quando se pretende incorporar um esquema num sistema operacional que possibilite a real manipulação de ficheiros. Estes tópicos dizem respeito a questões como o local de alojamento de ficheiros e a forma como as operações sobre ficheiros se podem coordenar com as operações dos esquemas de CP, a necessidade de avaliar os custos financeiros, os elementos necessários para suportar operações de reindexação, o tratamento dos nomes originais dos ficheiros e, por fim, o suporte para utilização de múltiplos fornecedores de serviços.

A partir dos tópicos identificados foram propostas duas arquiteturas para aprofundar o estudo, uma delas a arquitetura cliente-servidor tradicionalmente considerada em trabalhos sobre CP e, uma segunda arquitetura cliente-serviço de armazenamento, que não utiliza recursos de computação na *cloud*. A partir das duas arquiteturas consideradas foram desenvolvidos três sistemas que incorporaram soluções técnicas para alguns dos tópicos e para os quais foram feitas avaliações de performance, custo e segurança. Os tópicos tratados na implementação dos sistemas foram o armazenamento e operações sobre ficheiros (com exceção da operação de edição de ficheiros), o custo da reindexação e libertação de espaço no índice cifrado e o tratamento dos nomes originais dos ficheiros.

As operações de reindexação foram implementadas e testadas de duas formas diferentes, verificando-se que a implementação que recorre às pesquisas sobre o índice para obtenção da informação apresenta tempos de execução muito superiores à implementação que faz um novo tratamento de inserção aos ficheiros que não foram eliminados do sistema, pelo que deve ser a solução escolhida para soluções reais.

O tratamento do nomes originais dos ficheiros revelou que esta operação tem de estar alinhada com as construções criptográficas dos esquemas de CP utilizados, pois as transformações feitas aos nomes originais têm de permitir a correção da execução do esquema. No caso específico dos sistemas desenvolvidos, o Sophos forçava a que esta transformação utilizasse 64 *bits* para identificar unicamente os ficheiros cifrados, o que levou à utilização de números aleatórios para nomes de armazenamento dos ficheiros.

No que respeita à performance, os testes efetuados mostraram que o SCA é mais rápido a realizar as operações de inserção, remoção, reindexação e pesquisa. O sistema com pior performance nas operações de inserção e remoção é o SAA, com tempos de execução elevados e que se revelam inadequados para um sistema real. Nas operações de pesquisa, o sistema com pior desempenho é o SAABC.

Foi elaborado o modelo de custos de cada um dos sistemas desenvolvidos e conclui-se que o SAABC é o que tem menos gastos de operação e manutenção mas, quando existe a necessidade de realizar muitas operações de pesquisa, a diferença de custos dos vários sistemas torna-se pouco significativa na fatura final dos serviços utilizados.

A análise de segurança permitiu verificar que a associação das operações com ficheiros às operações sobre os índices cifrados aumenta o *leakage* do esquema de CP, pois permite que o fornecedor de serviços faça o mapeamento das entradas do índice ao ficheiro, ou conjunto de ficheiros, inserido. Apesar deste aumento de *leakage* não influenciar as propriedades de segurança garantidas pelo SCA, no SAA impede que este sistema garanta a propriedade de *backward privacy*. No SAABC este problema não surge devido à possibilidade de utilização da uma parte do sistema como *buffer* para as entradas do índice e pelo facto de se conseguir controlar quando as entradas no *buffer* são enviadas para o serviço de armazenamento utilizado. Este sistema beneficia ainda da utilização de uma *cache* do SGBD RocksDB-Cloud e pela forma como este faz a leitura dos ficheiros que contêm as entradas dos índices no serviço de armazenamento. Todas estas características e forma de funcionamento permitem que o SAABC garanta a propriedade de *backward privacy* e revele menos informação que o Tipo I, designado por *Backward Privacy with Insertion Pattern*. Conclui-se assim que, dos três sistemas desenvolvidos, o SAABC é o que menos informações dá a conhecer ao fornecedor de serviços sendo, por isso, o mais seguro.

Desta forma, é possível afirmar que o SCA apresenta vantagem se a performance for um fator importante para o utilizador do sistema e caso não seja necessário ter o serviço constantemente em funcionamento, o que permitirá desligar a VM onde o servidor está em execução. No entanto, se a segurança do sistema e disponibilidade dos recursos forem requisitos do utilizador, e sendo aceitável maiores tempos de execução das operações, o SAABC apresenta-se como uma solução mais adequada pois garante mais segurança e permite ter os recursos disponíveis 24 horas a baixo custo.

Por último, considera-se terem sido atingidos os objetivos desta dissertação. Foram identificadas as questões deixadas em aberto pelos trabalhos académicos analisados na área da CP e desenvolvidos sistemas onde as propostas de solução foram implementadas e testadas, com exceção do suporte para edição de ficheiros e utilização de múltiplas

soluções de armazenamento de diferentes fornecedores. O trabalho de investigação e de implementação dos sistemas permitiu responder à questão inicial desta dissertação, identificando os problemas e desenvolvendo as soluções para a real manipulação de ficheiros em conjunto com as operações do esquema de CP. As diferentes arquiteturas propostas permitiram mostrar que é possível reduzir os custos dos sistemas e que é possível diminuir o leakage do próprio esquema de CP utilizado.

7.1 Questões para trabalhos futuros

No âmbito desta dissertação foram desenvolvidos três sistemas que permitem a realização de operações de inserção, remoção e pesquisa sobre ficheiros. No entanto existem aspetos que podem ser melhorados e funcionalidades novas que podem ser implementadas:

Suporte para edição de ficheiros A edição é uma funcionalidade importante em sistemas que lidem com ficheiros. Foram identificados no capítulo 3 os problemas que esta operação coloca aos esquemas de CP analisados e que devem ser considerados em futuros trabalhos.

Suporte para utilização de Múltiplas Soluções de Armazenamento A capacidade de um sistema suportar dois ou mais fornecedores de serviços em simultâneo permite que o cliente possa alterar a localização dos ficheiros cifrados, dando mais flexibilidade e liberdade de escolha. Permite ainda que a informação armazenada possa ser distribuída, evitando a concentração da informação.

Bibliografia

[1] M. Armbrust, A. Fox, and R. Griffith. "Above the clouds: A Berkeley view of cloud computing". Em: *University of California, Berkeley, Tech. Rep. UCB* (2009), pp. 07–013. ISSN: 00010782. DOI: 10.1145/1721654.1721672. arXiv: 05218657199780521865715. URL: http://scholar.google.com/scholar?q=intitle:Above+the+clouds:+A+Berkeley+view+of+cloud+computing{\#}0.

[2] R. Bost. "$\Sigma o \phi o \zeta$ - Forward secure searchable encryption". Em: *Proceedings of the ACM Conference on Computer and Communications Security* 24-28-Octo (2016), pp. 1143–1154. ISSN: 15437221. DOI: 10.1145/2976749.2978303.

[3] Y. Zhang, J. Katz e C. Papamanthou. "All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption". Em: *Usenix Security* (2016), pp. 1–21.

[4] R. Curtmola e J Garay. "Searchable symmetric encryption: improved definitions and efficient constructions_2011version". Em: *Proceedings of the 13th ...* (2011), pp. 1–33. URL: http://dl.acm.org/citation.cfm?id=1180417.

[5] P.-A. F. Raphaël Bost. "Verifiable Dynamic Symmetric Searchable Encryption Optimality and Forward Security". Em: *IACR Cryptology ePrint Archive* (2016), pp. 1–40.

[6] D. Cash, P. Grubbs, J. Perry e T. Ristenpart. "Leakage-abuse attacks against searchable encryption". Em: *Proceedings of the ACM Conference on Computer and Communications Security* 2015-October (2015), pp. 668–679. ISSN: 15437221. DOI: 10.1145/2810103.2813700.

[7] P. Mell e T. Grance. "The NIST definition of cloud computing". Em: *Cloud Computing and Government: Background, Benefits, Risks*. 2011, pp. 171–173. ISBN: 9781617617843. DOI: 10.1016/b978-0-12-804018-8.15003-x. URL: https://www.nist.gov/publications/nist-definition-cloud-computing.

[8] "A view of cloud computing". Em: *Communications of the ACM* 53.4 (2010), pp. 50–58. ISSN: 00010782. DOI: 10.1145/1721654.1721672.

[9] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. M. Carreira, K. Krauth, N. Yadwadkar, J. Gonzalez, R. A. Popa, D. A. Patterson,

J. Carreira e J. E. Gonzalez. *Cloud Programming Simplified: A Berkeley View on Serverless Computing*. Rel. téc. 2019. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.html.

[10] *Computação sem servidor – Amazon Web Services*. URL: https://aws.amazon.com/pt/serverless/ (acedido em 09/01/2020).

[11] *Computação de Borda | CDN, Código Global Sem Servidor, Distribuição | AWS Lambda@Edge*. URL: https://aws.amazon.com/pt/lambda/edge/ (acedido em 09/01/2020).

[12] *AWS IoT Greengrass – Amazon Web Services*. URL: https://aws.amazon.com/pt/greengrass/ (acedido em 09/01/2020).

[13] D. X. Song, D. Wagner e A. Perrig. "Practical techniques for searches on encrypted data". Em: *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy* (2000), pp. 44–55. ISSN: 10637109. DOI: 10.1109/secpri.2000.848445.

[14] E. Stefanov, C. Papamanthou e E. Shi. "Practical Dynamic Searchable Encryption with Small Leakage". Em: February (2014), pp. 23–26. DOI: 10.14722/ndss.2014.23298.

[15] J. G. Chamani, D. Papadopoulos, C. Papamanthou e R. Jalili. "New constructions for forward and backward private symmetric searchable encryption". Em: *Proceedings of the ACM Conference on Computer and Communications Security*. Association for Computing Machinery, out. de 2018, pp. 1038–1055. ISBN: 9781450356930. DOI: 10.1145/3243734.3243833.

[16] S. Kamara, C. Papamanthou e T. Roeder. "Dynamic searchable symmetric encryption". Em: *Proceedings of the ACM Conference on Computer and Communications Security* (2012), pp. 965–976. ISSN: 15437221. DOI: 10.1145/2382196.2382298.

[17] S. Kamara e C. Papamanthou. "Parallel and dynamic searchable symmetric encryption". Em: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7859 LNCS (2013), pp. 258–274. ISSN: 03029743. DOI: 10.1007/978-3-642-39884-1_22.

[18] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu e M. Steiner. "Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation". Em: (2014), pp. 1–32. DOI: 10.14722/ndss.2014.23264.

[19] M. S. Islam, M. Kuzu e M. Kantarcioglu. "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation". Em: *Ndss* 20 (2012), pp. 1–15.

[20] R. Bost, B. Minaud e O. Ohrimenko. "Forward and backward private searchable encryption from constrained cryptographic primitives". Em: *Proceedings of the ACM Conference on Computer and Communications Security* (2017), pp. 1465–1482. ISSN: 15437221. DOI: 10.1145/3133956.3133980.

[21] Y. C. Chang e M. Mitzenmacher. "Privacy preserving keyword searches on remote encrypted data". Em: *Lecture Notes in Computer Science*. Vol. 3531. 2005, pp. 442–455. DOI: 10.1007/11496137_30.

[22] *What Is Amazon DynamoDB? - Amazon DynamoDB*. URL: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html (acedido em 08/01/2020).

[23] *Azure Cosmos DB | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/azure/cosmos-db/ (acedido em 08/01/2020).

[24] *Data model | Firestore | Google Cloud*. URL: https://cloud.google.com/firestore/docs/data-model (acedido em 10/01/2020).

[25] *Cloud Datastore | Google Cloud*. URL: https://cloud.google.com/datastore/ (acedido em 09/01/2020).

[26] *Redis*. URL: https://redis.io/documentation (acedido em 10/01/2020).

[27] *Amazon ElastiCache – Datastore e cache na memória*. URL: https://aws.amazon.com/pt/elasticache/ (acedido em 10/01/2020).

[28] *Azure Cache for Redis | Microsoft Azure*. URL: https://azure.microsoft.com/en-in/services/cache/ (acedido em 10/01/2020).

[29] *Cloud Memorystore for Redis documentation | Memorystore for Redis | Google Cloud*. URL: https://cloud.google.com/memorystore/docs/redis/ (acedido em 10/01/2020).

[30] *RocksDB | A persistent key-value store | RocksDB*. URL: https://rocksdb.org/ (acedido em 10/01/2020).

[31] *Home · facebook/rocksdb Wiki · GitHub*. URL: https://github.com/facebook/rocksdb/wiki (acedido em 10/01/2020).

[32] *GitHub - rockset/rocksdb-cloud: A library that provides an embeddable, persistent key-value store for fast storage optimized for AWS*. URL: https://github.com/rockset/rocksdb-cloud (acedido em 12/05/2020).

[33] *What is Amazon S3? - Amazon Simple Storage Service*. URL: https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html (acedido em 10/01/2020).

[34] *Quickstart: Azure Blob storage library v12 - .NET | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-dotnet (acedido em 10/01/2020).

[35] *Cloud Storage documentation | Cloud Storage | Google Cloud*. URL: https://cloud.google.com/storage/docs/ (acedido em 13/01/2020).

[36] P. Xu, S. Liang, W. Wang, W. Susilo, Q. Wu e H. Jin. "Dynamic searchable symmetric encryption with physical deletion and small leakage". Em: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10342 LNCS (2017), pp. 207–226. ISSN: 16113349. DOI: 10.1007/978-3-319-60055-0_11.

[37] M. Etemad, A. Küpçü, C. Papamanthou e D. Evans. "Efficient Dynamic Searchable Encryption with Forward Privacy". Em: *Proceedings on Privacy Enhancing Technologies* 2018.1 (2018), pp. 5–20. DOI: 10.1515/popets-2018-0002. arXiv: 1710.00208.

[38] *OpenSSE*. URL: https://opensse.github.io/ (acedido em 15/01/2020).

[39] *gRPC – A high-performance, open source universal RPC framework*. URL: https://grpc.io/ (acedido em 03/04/2020).

[40] *AEAD constructions - libsodium*. URL: https://doc.libsodium.org/secret-key{_}cryptography/aead (acedido em 07/04/2020).

[41] *Protocol Buffers | Google Developers*. URL: https://developers.google.com/protocol-buffers (acedido em 07/04/2020).

[42] *BLAKE2*. URL: https://blake2.net/ (acedido em 07/04/2020).

[43] *relic-toolkit · GitHub*. URL: https://github.com/relic-toolkit (acedido em 12/05/2020).

[44] *GitHub - sgtpepperpt/BISEN*. URL: https://github.com/sgtpepperpt/BISEN (acedido em 12/05/2020).

[45] *AWS SDK for C++*. URL: https://aws.amazon.com/pt/sdk-for-cpp/ (acedido em 12/05/2020).

[46] *A Glimpse into the World of Embedded Database Feat. RocksDB*. URL: https://medium.com/walmartlabs/https-medium-com-kharekartik-rocksdb-and-embedded-databases-1a0f8e6ea74f (acedido em 29/05/2020).

[47] *XChaCha20-Poly1305 construction - libsodium*. URL: https://doc.libsodium.org/secret-key{_}cryptography/aead/chacha20-poly1305/xchacha20-poly1305{_}construction (acedido em 07/04/2020).

[48] A. Zúquete. *Segurança em Redes Informáticas*. FCA - Editora de Informática, Lda, 2018.

[49] *Amazon Elastic Block Store (EBS) - Amazon Web Services*. URL: https://aws.amazon.com/ebs/ (acedido em 24/01/2020).

[50] *Pricing*. URL: https://aws.amazon.com/pricing/?nc2=h{_}ql{_}pr{_}ln (acedido em 24/01/2020).

ANEXO I

Anexo 1 Pseudocódigo Analisado

$\underline{\mathsf{Setup}()}$

1: $K_S \xleftarrow{\$} \{0,1\}^\lambda$
2: $(\mathsf{SK}, \mathsf{PK}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
3: $\mathbf{W}, \mathbf{T} \leftarrow$ empty map
4: **return** $((\mathbf{T}, \mathsf{PK}), (\mathbf{K_S}, \mathsf{SK}), \mathbf{W})$

$\underline{\mathsf{Search}(w, \sigma; \mathsf{EDB})}$

Client:
1: $K_w \leftarrow F_{K_S}(w)$
2: $(ST_c, c) \leftarrow \mathbf{W}[w]$ ▷ $c = n_w - 1$
3: **if** $(ST_c, c) = \perp$
4: **return** $\emptyset$
5: Send (K_w, ST_c, c) to the server.
Server:
6: **for** $i = c$ **to** 0 **do**
7: $UT_i \leftarrow H_1(K_w, ST_i)$
8: $e \leftarrow \mathbf{T}[UT_i]$
9: $\mathsf{ind} \leftarrow e \oplus H_2(K_w, ST_i)$
10: Output each ind
11: $ST_{i-1} \leftarrow \pi_{\mathsf{PK}}(ST_i)$
12: **end for**

$\underline{\mathsf{Update}(\mathsf{add}, w, \mathsf{ind}, \sigma; \mathsf{EDB})}$

Client:
1: $K_w \leftarrow F(K_S, w)$
2: $(ST_c, c) \leftarrow \mathbf{W}[w]$
3: **if** $(ST_c, c) = \perp$ **then**
4: $ST_0 \xleftarrow{\$} \mathcal{M}$, $c \leftarrow -1$
5: **else**
6: $ST_{c+1} \leftarrow \pi^{-1}_{\mathsf{SK}}(ST_c)$
7: **end if**
8: $\mathbf{W}[w] \leftarrow (ST_{c+1}, c+1)$
9: $UT_{c+1} \leftarrow H_1(K_w, ST_{c+1})$
10: $e \leftarrow \mathsf{ind} \oplus H_2(K_w, ST_{c+1})$
11: Send (UT_{c+1}, e) to the server.
Server:
12: $\mathbf{T}[UT_{c+1}] \leftarrow e$

Figura I.1: Pseudo-código do esquema sophos-b [2]

Setup()

1: $K_\Sigma \xleftarrow{\$} \{0,1\}^\lambda$, $\mathbf{W}, \mathsf{EDB} \leftarrow$ empty map
2: **return** $(\mathsf{EDB}, K_\Sigma, \mathbf{W})$

Search$(K_\Sigma, w, \sigma; \mathsf{EDB})$

Client:
1: $K_w || K'_w \leftarrow F_{K_\Sigma}(w)$, $c \leftarrow \mathbf{W}[w]$ ▷ $c = n_w - 1$
2: **if** $c = \perp$ **then return** $\emptyset$
3: $ST \leftarrow \widetilde{F}.Constrain(K_w, C_c)$ ▷ C_c *is the circuit evaluating to 1 on* $\{0, \ldots, c\}$
4: Send (K'_w, ST, c) to the server.
Server:
5: **for** $i = c$ **to** 0 **do**
6: $T_i \leftarrow \widetilde{F}(ST, i)$
7: $UT_i \leftarrow H_1(K'_w, T_i)$
8: $e \leftarrow \mathsf{EDB}[UT_i]$
9: $\mathsf{ind} \leftarrow e \oplus H_2(K'_w, T_i)$
10: Output each ind
11: **end for**

Update$(K_\Sigma, \mathsf{add}, w, \mathsf{ind}, \sigma; \mathsf{EDB})$

Client:
1: $K_w || K'_w \leftarrow F(K_\Sigma, w)$, $c \leftarrow \mathbf{W}[w]$
2: **if** $c = \perp$ **then** $c \leftarrow -1$
3: $T_w^{c+1} \leftarrow \widetilde{F}(K_w, c+1)$, $\mathbf{W}[w] \leftarrow c+1$
4: $UT_{c+1} \leftarrow H_1(K'_w, T_w^{c+1})$, $e \leftarrow \mathsf{ind} \oplus H_2(K'_w, T_w^{c+1})$
5: Send (UT_{c+1}, e) to the server.
Server:
6: $\mathsf{EDB}[UT_{c+1}] \leftarrow e$

Figura I.2: Pseudo-código esquema de Criptografia Simétrica Pesquisável (CSP) utilizando RCPRFs [20]

Setup()

1: $(\mathsf{EDB}_{\mathsf{add}}, K_{\mathsf{add}}, \sigma_{\mathsf{add}}) \leftarrow \Sigma_{\mathsf{add}}.\mathsf{Setup}()$
2: $(\mathsf{EDB}_{\mathsf{del}}, K_{\mathsf{del}}, \sigma_{\mathsf{del}}) \leftarrow \Sigma_{\mathsf{del}}.\mathsf{Setup}()$
3: $K_{tag}, K_S \leftarrow \{0,1\}^\lambda$, $\mathbf{PSK}, \mathbf{SC}, \mathsf{EDB}_{cache} \leftarrow$ empty map
4: **return** $((\mathsf{EDB}_{\mathsf{add}}, \mathsf{EDB}_{\mathsf{del}}, \mathsf{EDB}_{cache}), (K_{\mathsf{add}}, K_{\mathsf{del}}, K_{tag}, K_S), (\sigma_{\mathsf{add}}, \sigma_{\mathsf{del}}, \mathbf{PSK}, \mathbf{SC}))$

Search$(K_\Sigma, w, \sigma; \mathsf{EDB})$

Client:
1: $i \leftarrow \mathbf{SC}[w]$.
2: **if** $i = \perp$ **return** $\emptyset$
3: Send $sk_0 = \mathbf{PSK}[w]$ to the server.
4: $\mathbf{PSK}[w] \leftarrow \mathsf{PPKE.KeyGen}(1^\lambda)$, $\mathbf{SC}[w] \leftarrow i+1$.
5: Send $\mathsf{tkn} \leftarrow F(K_S, w)$ to the server.
Client (C) & Server (S):
6: C and S run $\Sigma_{\mathsf{add}}.\mathsf{Search}(K_{\mathsf{add}}, w||i, \sigma_{\mathsf{add}}; \mathsf{EDB}_{\mathsf{add}})$. The server gets a list $((ct_1, t_1^{\mathsf{add}}), \ldots, (ct_n, t_n^{\mathsf{add}}))$ of ciphertexts and tags.
7: C and S run $\Sigma_{\mathsf{del}}.\mathsf{Search}(K_{\mathsf{del}}, w||i, \sigma_{\mathsf{del}}; \mathsf{EDB}_{\mathsf{del}})$. The server gets a list $((sk_1, t_1^{\mathsf{del}}), \ldots, (sk_m, t_m^{\mathsf{del}}))$ of key elements.
8: S decrypts the ciphertexts with $\mathsf{SK} = (sk_0, sk_1, \ldots, sk_m)$, and obtains the list $NewInd = ((\mathsf{ind}_1, t_1), \ldots, (\mathsf{ind}_\ell, t_\ell))$.
Server:
9: $OldInd \leftarrow \mathsf{EDB}_{cache}[\mathsf{tkn}]$
10: Remove from $OldInd$ the indices whose tags are in $\{t_j^{\mathsf{del}}\}$.
11: $Res \leftarrow OldInd \cup NewInd$, $\mathsf{EDB}_{cache}[\mathsf{tkn}] \leftarrow Res$
12: **return** Res

Update$(K_\Sigma, \mathsf{add}, w, \mathsf{ind}, \sigma; \mathsf{EDB})$

1: $t \leftarrow F_{K_{tag}}(w, \mathsf{ind})$
2: $sk_0 \leftarrow \mathbf{PSK}[w]$, $i \leftarrow \mathbf{SC}[w]$
3: **if** $sk_0 = \perp$ **then**
4: $sk_0 \leftarrow \mathsf{PPKE.KeyGen}(1^\lambda)$, $\mathbf{PSK}[w] \leftarrow sk_0$
5: $i \leftarrow 0$, $\mathbf{SC}[w] \leftarrow i$
6: **end if**
7: **if** $\mathsf{op} = \mathsf{add}$ **then**
8: $ct \leftarrow \mathsf{PPKE.Encrypt}(sk_0, \mathsf{ind}, t)$
9: Run $\Sigma_{\mathsf{add}}.\mathsf{Update}(K_{\mathsf{add}}, \mathsf{add}, w||i, (ct, t), \sigma_{\mathsf{add}}; \mathsf{EDB}_{\mathsf{add}})$
10: **else** ▷ $\mathsf{op} = \mathsf{del}$
11: $(sk'_0, sk_t) \leftarrow \mathsf{PPKE.IncPuncture}(sk_0, t)$
12: Run $\Sigma_{\mathsf{del}}.\mathsf{Update}(K_{\mathsf{del}}, \mathsf{add}, w||i, (sk_t, t), \sigma_{\mathsf{del}}; \mathsf{EDB}_{\mathsf{del}})$
13: $\mathbf{PSK}[w] \leftarrow sk'_0$
14: **end if**

Figura I.3: Pseudo-código do esquema Janus [20]

$(\mathcal{I}'_c, C_{w_i})(\mathcal{I}'_s) \leftarrow \mathtt{Search}(sk, \mathcal{I}_c, w)(\mathcal{I}_s, C)$:

Client:

1: $K_w = G(K_G, w||\mathsf{SearchCnt}[w])$
2: Send the token $(K_w, cnt = \mathtt{FileCnt}[w])$ to the server.

Server:

3: $F_w = \{\}$
4: **for** $i = 1$ *to cnt* **do** ▷ No. of files w appears in.
5: $id(f_i) = \mathsf{DictW}[h(K_w, i||0)] \oplus h(K_w, i||1)$
6: $F_w = F_w \cup \{id(f_i)\}$
7: Delete $\mathsf{DictW}[h(K_w, i||0)]$ ▷ Delete accessed entry.
8: Send all files corresponding to F_w to the client.

Client:

9: Decrypt and consume the received files.
10: $\mathsf{SearchCnt}[w]$++ ▷ Searched for w once more.
11: $K'_w = G(K_G, w||\mathsf{SearchCnt}[w])$; ▷ A fresh key for w.
12: WPairs = { }
13: **for** $i = 1$ *to cnt* **do**
14: $\text{addrW}_i = h(K'_w, i||0)$;
15: $\text{valW}_i = h(K'_w, i||1) \oplus id(f_i)$
16: WPairs = WPairs $\cup$ {(addrW$_i$, valW$_i$)}
17: Upload WPairs to the server.
18: The server adds WPairs into DictW.

Figura I.4: Pseudo-código do esquema de Etemad et al. [37]

$\underline{\mathsf{Setup}(DB)}$:

1: $\Sigma.\mathsf{Setup}(\mathsf{DB})$, $K_\Sigma \xleftarrow{\$} \{0,1\}^\lambda$

$\underline{\mathsf{Search}(K_\Sigma, w, \sigma; \mathsf{EDB})}$

1: Client and Server run $\Sigma.\mathsf{Search}(w)$, the client gets the list of results R.

Client:

2: $K_w \leftarrow F(K_\Sigma, w)$
3: Decrypt R as $(E_{K_w}(\mathsf{ind}_1, \mathsf{op}_1), \ldots, E_{K_w}(\mathsf{ind}_n, \mathsf{op}_n))$
4: Return $\{\mathsf{ind} : \exists i, (\mathsf{ind}_i, \mathsf{op}_i) = (\mathsf{ind}, \mathsf{add}) \wedge \forall j > i, (\mathsf{ind}_j, \mathsf{op}_j) \neq (\mathsf{ind}, \mathsf{del})\}$

$\underline{\mathsf{Update}(K_\Sigma, \mathsf{add}, w, \mathsf{ind}, \sigma; \mathsf{EDB})}$

1: Client: $K_w \leftarrow F(K_\Sigma, w)$
2: Client and Server run $\Sigma.\mathsf{Update}(\mathsf{add}, w, E_{K_w}(\mathsf{ind}, \mathsf{op}))$

Figura I.5: Pseudo-código do esquema genérico B(Σ) [20]

Client:
1: **if FileCnt**$[w]$ is NULL **then**
2: **FileCnt**$[w] = 0$
3: **end if**
4: **FileCnt**$[w]$++
5: $addr = G_K(w, \textbf{FileCnt}[w]||0)$
6: $val = (id||op) \oplus G_K(w, \textbf{FileCnt}[w]||1)$
7: Send $(addr, val)$ to the server
Server:
8: Set **DictW**$[addr] = val$

Figura I.6: Pseudo-código do algoritmo de *update* do esquema Mitra [15]

Client:
1: TList = { }
2: **for** $i = 1$ to **FileCnt**$[w]$ **do**
3: $T_i = G_K(w, i||0)$
4: TList = TList $\cup \{T_i\}$
5: **end for**
6: Send TList to the server
Server:
7: $F_w = \{\}$
8: **for** $i = 1$ to TList.size **do**
9: $F_w = F_w \cup$ **DictW**[TList$[i]$]
10: **end for**
11: Send F_w to the client
Client:
12: $R_w = \{\}$
13: **for** $i = 1$ to $F_w.size$ **do**
14: $(id||op) = F_w[i] \oplus G_K(w, i||1)$
15: $R_w = R_w \cup (id||op)$
16: **end for**
17: Remove ids that have been deleted from R_w
18: **return** R_w

Figura I.7: Pseudo-código do algoritmo de *search* do esquema Mitra [15]

$\underline{\mathsf{Setup}(DB):}$

1: $\mathsf{T}[w] \leftarrow 0$ for all w, $K_\Sigma \xleftarrow{\$} \{0,1\}^\lambda$
2: $\mathsf{DB}' \leftarrow \mathsf{DB}$ where keywords w are replaced by $w||0$
3: $\Sigma.\mathsf{Setup}(\mathsf{DB}')$

$\underline{\mathsf{Search}(K_\Sigma, w, \sigma; \mathsf{EDB})}$

1: Client: $K_w \leftarrow H(K_\Sigma, w, \mathsf{T}[w])$
2: Client and Server run $R \leftarrow \Sigma.\mathsf{Search}(w||\mathsf{T}[w])$ ▷ *Server can erase all retrieved elements from memory*
Client:
3: Decrypt R as $(E_{K_w}(\mathsf{ind}_1, \mathsf{op}_1), \ldots, E_{K_w}(\mathsf{ind}_n, \mathsf{op}_n))$
4: $R' \leftarrow \{\mathsf{ind} : \exists i, (\mathsf{ind}_i, \mathsf{op}_i) = (\mathsf{ind}, \mathsf{add}) \ \wedge \forall j > i, (\mathsf{ind}_j, \mathsf{op}_j) \neq (\mathsf{ind}, \mathsf{del})\}$
5: Send R' to Server
6: $\mathsf{T}[w] \leftarrow \mathsf{T}[w] + 1$
7: **for all** $\mathsf{ind} \in R'$ **do** ▷ *In parallel*
8: Run $\mathsf{Update}(K_\Sigma, \mathsf{add}, w, \mathsf{ind}, \sigma; \mathsf{EDB})$
9: **end for**

$\underline{\mathsf{Update}(K_\Sigma, \mathsf{add}, w, \mathsf{ind}, \sigma; \mathsf{EDB})}$

1: Client: $K_w \leftarrow H(K_\Sigma, w, \mathsf{T}[w])$
2: Client and Server run $\Sigma.\mathsf{Update}(\mathsf{add}, w||\mathsf{T}[w], E_{K_w}(\mathsf{ind}, \mathsf{op}))$

Figura I.8: Pseudo-código do esquema genérico B'(Σ) [20]

```
1:  mapKey = (w, id)
2:  (rootID′, updt_cnt) ← OMAP_upd.Find(mapKey, rootID′)
3:  if op = add then
4:      if updt_cnt = NULL or updt_cnt = −1 then
5:          if UpdtCnt[w] is NULL then
6:              UpdtCnt[w] = 0
7:          end if
8:          UpdtCnt[w]++
9:          data = ((w, id), UpdtCnt[w])
10:         rootID′ ← OMAP_upd.Insert(data, rootID′)
11:         data = ((w, UpdtCnt[w]), id)
12:         rootID ← OMAP_src.Insert(data, rootID)
13:         LastInd[w] = id
14:     end if
15: else if op = del then
16:     if updt_cnt > 0 then
17:         data = (mapKey, −1)
18:         rootID′ ← OMAP_upd.Insert(data, rootID′)
19:         UpdtCnt[w]--
20:         if UpdtCnt[w] > 0 then      ▷ There are entries for w
21:             if UpdtCnt[w] + 1 ≠ updt_cnt then
22:                 data = ((w, LastInd[w]), updt_cnt)
23:                 rootID′ ← OMAP_upd.Insert(data, rootID′)
24:                 data = ((w, updt_cnt), LastInd[w])
25:                 rootID ← OMAP_src.Insert(data, rootID)
26:             end if
27:             key = (w, UpdtCnt[w])
28:             (rootID, lastID) ← OMAP_src.Find(key, rootID)
29:             LastInd[w] = lastID
30:         else
31:             LastInd[w] = 0
32:         end if
33:     end if
34: end if
35: Execute necessary dummy oblivious map accesses
```

Figura I.9: Pseudo-código da operação de atualização do esquema Orion [15]

1: $R = \{\}$
2: **for** $i = 1$ **to UpdtCnt**$[w]$ **do** ▷ Execute in batch
3: $(rootID, id) \leftarrow OMAP_{src}.Find((w, i), rootID)$
4: $R = R \cup \{id\}$
5: **end for**
6: **return** R

Figura I.10: Pseudo-código da operação de pesquisa do esquema Orion [15]